ISSN 0868-6157

КОМПЬЮТЕР ПРЕСС

7'93

КОМПЬЮТЕР ПРЕСС

7'93



КОМПЬЮТЕРПРЕСС

Издается с 1989 года
Выходит 12 раз в год
7'93 (43)

Адрес редакции:
113093 Москва, аб.ящик 37
Факс: (095) 470-31-05
Телефон для справок: (095) 471-32-63
Отдел рекламы: (095) 470-31-05
E-mail: editorial@computerpress.msk.su

Реклама в номере:

Ответственность за информацию, приведенную в рекламных материалах, несет рекламодатель.

Сдано в набор 14.05.93. Подписано к печати 15.06.93. Формат 84x108/16. Печать офсетная. Бумага типографская. Усл.печ.листов 8,4 + 0,42 (обложка). Кр.-отт. 10,08.Тираж 52000 экз. Заказ 3781. С-19.

Оригинал-макет подготовлен фирмой «КомпьютерПресс».

Тексты проверены системой «ОРФО».

Отпечатано в полиграфической фирме «Красный пролетарий» РГИИЦ «Республика». 103473 Москва, И-473, Краснопролетарская, 16.

Языки программирования

В настоящее время насчитывается более сотни языков программирования — средств для управления компьютерами. 30-40-летняя история языков программирования характеризовалась всплесками, которые приходились на появление новых концепций: структурное программирование, объектно-ориентированное и так далее.

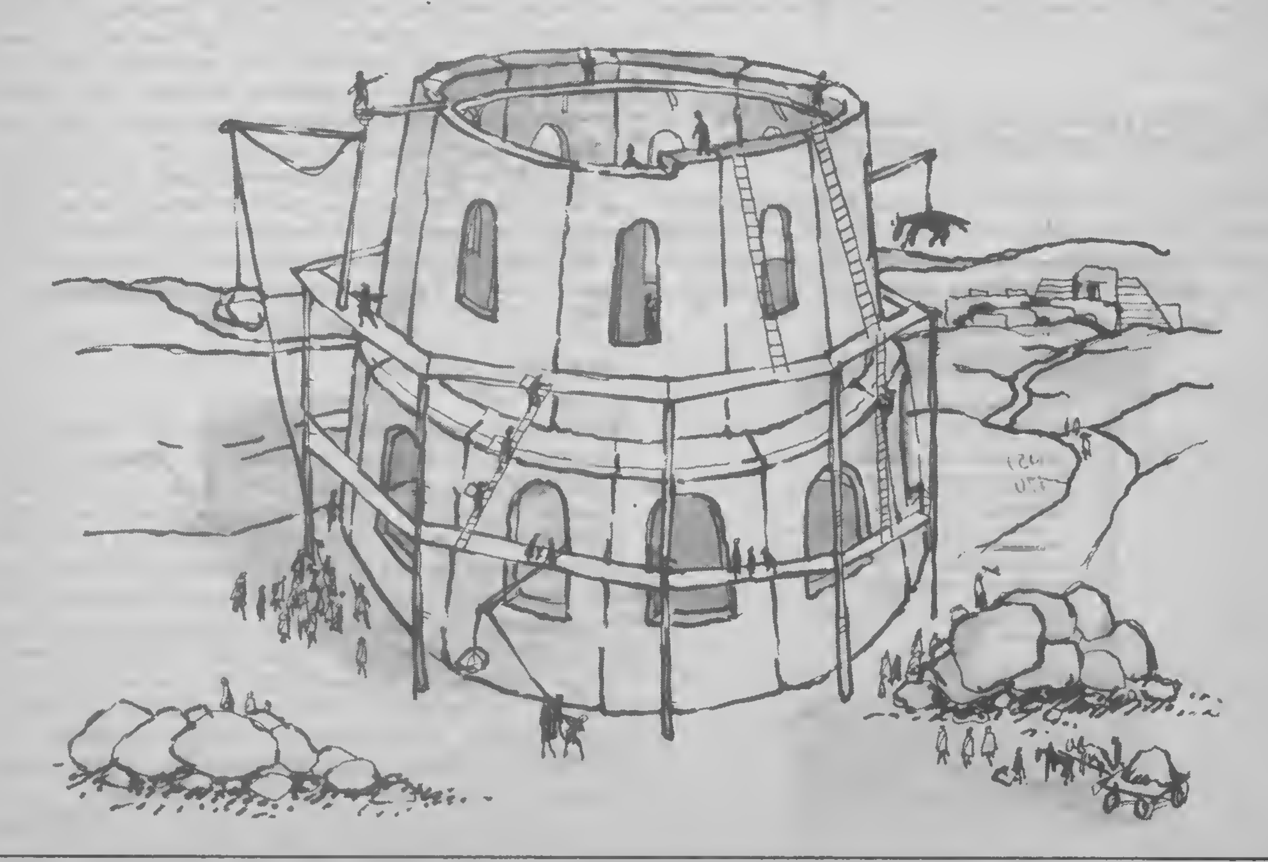
Если мы рассмотрим языки, используемые на персональных компьютерах, то увидим, что несомненным лидером здесь является C/C++ — этот язык выбран в качестве "индустриального стандарта" большинством разработчиков программного обеспечения. К сожалению, объектно-ориентированное расширение этого языка сделало его синтаксически более громоздким, а ряд его реализаций (средствами компиляторов) полностью снял вопрос о переносимости. Но между тем, порядка 80% программного обеспечения создается именно на этом языке. Какие при этом используются компиляторы — уже другой вопрос.

Объектно-ориентированное расширение языка Pascal, реализованное фирмой Borland, нашло очень много приверженцев и является не только средством для изучения объектно-ориентированного программирования, но и хорошим инструментом для создания прикладных программ. Там, где не хватает гибкости и мощности языков высокого уровня, используется язык ассемблера (даже в программах на C/C++, который иногда называют "ассемблером высокого уровня").

Язык Basic, изначально создававшийся как средство для обучения, давно уже перерос это качество и вполне может использоваться для создания серьезных приложений. Интересно отметить, что ситуацию с языками программирования (их популярностью) очень четко отражает рынок программного обеспечения: предлагается более десятка компиляторов C/C++, 2-3 компилятора языка Pascal, 1-2 компилятора языка Basic. Другие языки представлены одним, иногда двумя компиляторами, так же как и Windows-версии ряда языков.

Давайте кратко рассмотрим эволюцию языков программирования — ведь знание истории поможет нам понять будущее.

Языки программирования появились около 40 лет назад. Сегодня можно выделить 4 основных поколения языков:



- первое (конец 50-х): языки Fortran и Algol;
- второе (середина 60-х): Cobol, Lisp;
- третье (70-е годы): PL/1, Pascal, Simula;
- четвертое (80-е годы): Smalltalk, Object Pascal, C++, Ada.

В каждой группе представлены лишь те языки, которые "пережили время" и используются до сих пор. В 70-80 годы возникло множество языков, большинство из которых оказались нежизнеспособными.

К структурному программированию

Первые языки создавались чисто для решения конкретных задач: описания и вычисления формул, экономических или инженерных расчетов. По мере развития аппаратных средств, снижения их себестоимости, а также расширения сферы применения компьютеров эти языки расширялись функциями ввода/вывода, поддержки файловой системы, взаимодействия с операционной системой и т.д. По мере усложнения задач, решавшихся с помощью компьютеров, расширялись возможности и самих языков. Появилась поддержка подпрограмм: основная программа состояла из данных и набора подпрограмм, обрабатывающих эти данные. Поддержка подпрограмм и, как следствие, механизма передачи параметров явилась отправной точкой для реализации методологии структурного программирования, которая стала доминирующей в конце 70-х — начале 80-х годов.

От структурного программирования к объектно-ориентированному

Реализация вложенности подпрограмм и ограничение области действия различных компонентов программы, а также возможность создания больших программ на основе подпрограмм привели к изменению в архитектуре самих языков. Разработка больших программных проектов потребовала иного подхода к компоновке программ, в результате появились механизмы раздельной компиляции. Появилось понятие модульности и, как следствие — языки, поддерживающие модульность и раздельную компиляцию (см. статью "Почему я работаю на языке Модула-2"). Абстракция данных, типизация и модульность стали основами технологии объектно-ориентированного программирования, которая уверенно лидирует в современном компьютерном мире. Причем наиболее популярны не малоизвестные новые языки, а объектно-ориентированные расширения уже знакомых пользователю языков, таких как C или Pascal.

От объектно-ориентированного программирования к визуальному

Рост популярности среды Microsoft Windows вызвал к жизни целый класс языков, ориентированных на разработку приложений для этой среды. Удачны или нет те или иные реализации, покажет время, но с появлением таких языков, как Visual Basic и Actor, стало возможным говорить о технологии визуального, наглядного программирования. С одной стороны, эта технология значительно сокращает цикл разработки программного обеспечения, однако с другой — лишает его индивидуальности, а значит, и творческой стороны. Всевозможные конструкторы, автоматические генераторы кодов и прочие средства практически не требуют от программиста каких-либо дополнительных действий. Здесь есть крайности: кажущаяся простота создания приложения на самом деле оборачивается кропотливой работой и неудовлетворительным результатом (см. статью "Actor 4.0 Professional. Программирование в среде Windows"). Средства визуального программирования появляются и в среде DOS; в текстовом режиме достаточно сложно сделать что-либо пристойное и удобное на эту тему, но то, что уже реализовано, вселяет надежду.

Что нас ждет?

На мой взгляд, в течение нескольких лет никаких существенных изменений в технологии программирования и в самих языках произойти не должно. Существующие технологии еще не в полном объеме исчерпали себя. В то же время наметилась тенденция к упрощению самого процесса создания программ: уже есть программные продукты, которые позволяют создавать приложения без программирования. Функциональный уровень таких приложений пока достаточно ограничен. Таким образом, развитие языков программирования видится в комбинации объектной ориентированности с наглядностью. Хотя, кто знает, какие сюрпризы готовит нам этот бурно развивающийся мир программных средств. Поживем — увидим.

А.Федоров

ACTOR 4.0 Professional. Программирование в среде Windows

Эволюция графической среды Microsoft Windows изменила не только мышление пользователей, но и сами средства, при помощи которых создаются Windows-приложения. Среда Microsoft Windows представляет собой привлекательный для пользователя графический интерфейс, удобный в использовании и изучении, тогда как создание Windows-приложений до недавнего времени было довольно сложным процессом. Существовал всего один компилятор (Microsoft C), позволявший создавать Windows-приложения, и набор соответствующих утилит (Microsoft SDK). Постепенно, с приходом технологии объектно-ориентированного и визуального программирования, ситуация изменилась: в настоящее время Windows-приложения можно создавать и на C, и на Pascal, и даже на Basic. Уже больше не требуется постоянного перехода из DOS в Windows для того, чтобы откомпилировать программу, а затем посмотреть результаты — все компиляторы работают непосредственно в среде Windows.

Объектно-ориентированный подход позволил существенно упростить создание Windows-приложений. Пионером такого подхода к созданию приложений в среде Windows была фирма Whitewater Group (в настоящее время входящая в фирму Symantec), разработавшая в середине 80-х годов объектно-ориентированный язык Actor. Этот обзор посвящен версии 4.0 данного продукта, любезно предоставленной мне московским представительством фирмы Symantec.

Теория

Язык Actor, как и Smalltalk, — это чисто объектно-ориентированный язык. В нем существует только один тип переменной: объект. Объекты являются экземплярами классов. Actor содержит более 100 классов, включающих простые классы типа Number (число) или Char (символ) и более сложные, такие, например, как Array (массив) и Window (окно).

Для каждого класса определен ряд функций (называемых методами в терминологии ООП), которые задают свойства данного класса. Для вызова какого-либо метода объекту (экземпляру класса) посылается необходимое сообщение, содержащее название этого метода:

```
message(object, arg1, arg2)
```

Язык Actor поддерживает механизм наследования; для каждого класса существует родительский класс. Вместо множественного наследования, присущего таким языкам, как C++, в языке Actor реализован механизм протоколов. Протоколы — это объекты, похожие на классы, но существующие в отдельной иерархии. Как и классы, протоколы имеют методы, но в отличие от классов, протоколы не имеют экземпляров. Класс может "подсоединиться" к протоколу. В этом случае методы протокола будут доступны экземплярам класса. Одним словом, методы протокола — это шаблон для метода класса.

Переняв многое из концепции и реализации языка Smalltalk, синтаксически Actor тем не менее представляет собой нечто среднее между Pascal и C, больше приближаясь к языку Pascal. Таким образом, программисты, знакомые с тем или другим языком, смогут без труда освоить Actor.

Среди многочисленных классов, включенных в комплект поставки Actor, наиболее интересной является библиотека классов Object Windows (та самая, которая поставляется с компиляторами Turbo Pascal for Windows и Borland C++). Те, кто имел возможность испробовать эту библиотеку с одним из указанных компиляторов, не могут не согласиться, что с ее помощью создание Windows-программ становится несколько проще.

Для создания и редактирования ресурсов в комплект Actor Professional входит редактор ресурсов Whitewater Resource Toolkit (написанный на языке Actor). Одна из версий этого редактора поставлялась в комплекте с компилятором Turbo Pascal for Windows 1.0, но затем была заменена на собственную разработку фирмы Borland — Resource Workshop. Whitewater Resource Toolkit объединяет в единую среду редакторы для различных типов ресурсов и позволяет отказаться от ис-

пользования пакетного компилятора ресурсов фирмы Microsoft (RC.EXE) и ряда утилит, входящих в Windows SDK.

Из компонентов, включенных в комплект поставки Actor Professional, интересна также библиотека классов Object Graphics, которая выпускается и для компиляторов Turbo Pascal for Windows и Borland C++ (см. КомпьютерПресс № 3´93).

В комплект также входит библиотека для поддержки баз данных Q+E Database Library (QELIB) фирмы Pioneer Software Systems Inc. Эта библиотека позволяет выполнять SQL-запросы к различным базам данных. QELIB состоит из библиотеки классов Actor и набора DLL, поддерживающих следующие форматы: dBASE II, III и IV, Paradox (требуется наличие Paradox Engine), Excel и ASCII-файлы.

Практика

Как известно, практика является критерием истины. Поэтому познакомившись с возможностями языка Actor чисто теоретически, я рискнул попробовать на практике создать Windows-программу с помощью языка Actor. Я пытался решить простейшую задачу: в программе создается окно, в меню которого имеется команда Help|About;, при выборе этой команды отображается панель сообщения с каким-либо текстом. Приводимое ниже описание содержит шаги, предпринятые мной для создания соответствующего приложения. Если кто-либо хочет попробовать, необходимо выполнить перечисленные ниже шаги. Итак, как говорят фокусники, следите за руками.

Запускаем Actor. Это можно сделать с помощью иконки или командой

```
WIN ACTOR
```

1. Для простоты будем создавать окно на базе класса Window (напомню, что в Actor все является объектами). В меню Browse выбираем команду Classes и попадаем в браузер классов. Далее необходимо найти класс Window. Когда класс найден, создаем его наследника (Browser|Class|Make Descendant). Попадаем в панель описания класса.

В поле Name заносим название создаваемого класса, скажем Sample. После этого нажимаем кнопку Accept.

Итак, мы создали класс — наследник класса Window

```
/* Class Sample */!!
inherit(Window, #Sample, #(somevar), 2, nil)!!
now(class(Sample))!!
now(Sample)!!
```

2. Теперь необходимо заняться внешним видом окна. В среде Actor существует специальный макроязык — WDL (Window Description Language), с помощью которого окно описывается всего несколькими строками. Для моего примера такое описание выглядит следующим образом:

```
Window sample
  class: Window
  title: "Hello, Actor!"
Menu mainmenu
 Popup help text: "Help"
  Item about text: "About.."
 End
End
```

Описание вводится в редакторе, который появляется при выборе команды Browse|Attributes.

После того как описание введено, можно проверить внешний вид окна и функциональность меню — для этого используется команда DoIt! Когда внешний вид окна определен и меню описано и подключено, имеет смысл сохранить это описание в файле. Выбираем команду File|Save as и сохраняем описание как SAMPLE.WDL.

3. Теперь нам необходимо создать метод, который будет получать управление при выборе команды Help|About. Для этого возвращаемся в браузер классов (Browser|Class). Из меню Templates выбираем команду New Action. В результате мы получаем шаблон метода — обработчика события. Описываем наш метод следующим образом:

```
/*************************************
    Method to respond to Help|About
*************************************/
Action helpAbout(self, msg) #[#about]
{
errorBox("Actor 4.03","Sample Actor Application")
}!!
```

4. Для того чтобы данный метод подключить к классу, необходимо выполнить команду Compile!. Если это первый метод для данного класса (как в нашем случае), появится панель диалога, в которой необходимо указать, к какому классу относится данный метод. В нашем случае необходимо указать Descendant.

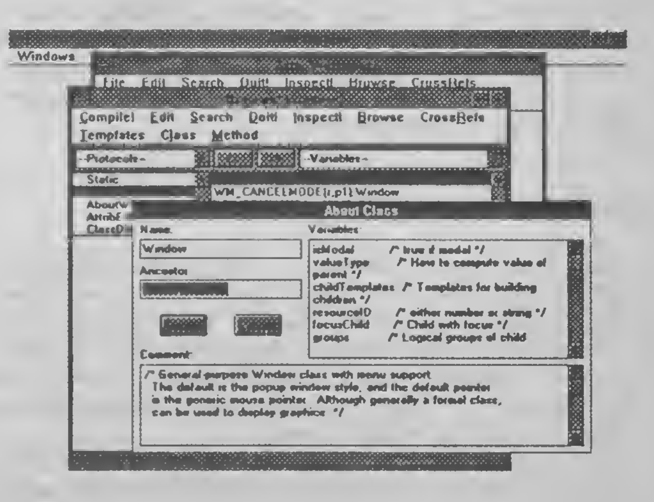


Рис. 1. Браузер классов

5. Отлично, меню работает (см. пункт 2), обработчик события создан и подключен к классу (пп. 3-4). Можно попробовать. Вернемся в окно, в котором мы описывали наше окно, и выполним команду DoIt!. А затем выполним команду Help|About. Панель сообщения появилась? Отлично. Мы практически создали первую Actor-программу.

6. Теперь нашу программу необходимо извлечь из среды Actor и превратить в независимую. Этот процесс может показаться несколько необычным, но так уж здесь устроено. Сначала необходимо создать еще один класс — класс-приложение. Этот класс должен быть наследником класса Application. Ищем этот класс в браузере классов (Browse|Class) и создаем его наследника (Class|Make Descendant). Имя нашего класса будет SampleApp. Нажмем кнопку Accept.

7. Теперь нам необходимо создать метод initMainWindow. Для этого выбираем команду Templates|New Method и вводим следующий текст:

```
/**************************************
       Sample Application Class
**************************************/!!
inherit(Application, #SampleApp, nil, 2, nil)!!
now(class(SampleApp))!!
now(SampleApp)!!
/* Sample Application Initialization */
Def initMainWindow(self)
{
 mainWindow := loadTop(sample);
}!!
```

С помощью метода loadTop основное окно нашего приложения создается на основе описания, сохраненного ранее в файле SAMPLE.WDL. Для подключения этого метода к классу необходимо выполнить команду Compile!

8. Если вы уже устали, то потерпите еще немного, еще несколько шагов — и мы у цели. В окне Actor Workspace мы можем проверить все произведенные выше действия. Введем следующие команды:

```
Sam := new(SampleApp);
init(Sam, nil);
```

9. Вся активность в среде Actor слегка загрязняет память самой среды. Для ее очистки используется команда Cleanup!. Затем необходимо сохранить образ среды в файле. Выполним команду File|Snapshot. Сохраним среду в файле SAMPLE.IMA. Затем нам необходимо очистить память от всяких разных классов, которые загружаются по умолчанию, но не используются нашим приложением. Команда Seal-Off позволяет выполнить такую очистку. В панели диалога необходимо указать название класса — SampleApp, имя файла с образом среды — SAMPLE.IMA и нажать кнопку Ok. Процесс очистки может быть не очень быстрым, но после него становится просторнее. На этом первый сеанс со средой Actor заканчивается.

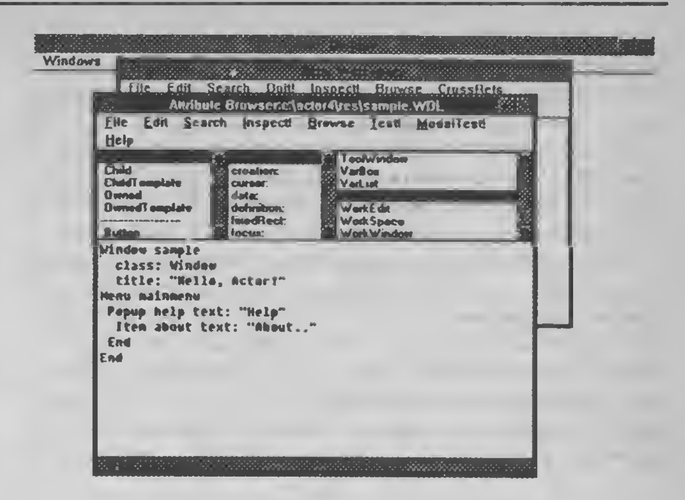


Рис. 2. Панель описания окна

10. В результате описанных действий мы получили образ нашего приложения — файл SAMPLE.IMA. Это приложение можно выполнить, если указать название файла при запуске Actor:

```
ACTOR SAMPLE.IMA
```

но нашей задачей является создание независимого от среды приложения. Следующий шаг — подключение ресурсов к файлу.

11. Для этого необходимо скопировать файл ACTOR.EXE в SAMPLE.EXE:

```
COPY ACTOR.EXE SAMPLE.EXE
```

и создать следующий файл описания ресурсов:

```
#include "style.h"
#include "actor.h"
sample WDL   sample.wdl
STRINGTABLE
BEGIN
 IDSNAME  "Sample"
 IDSAPP   "SAMPLE.IMA"
END
```

12. Откомпилируем файл описания ресурсов (sample.rc):

```
rc sample
```

С помощью компилятора ресурсов мы получим обновленную версию программы SAMPLE.EXE.

13. Уф! Готово! Мы имеем простейшее Windows-приложение, созданное с помощью языка Actor. Для работы данного приложения необходимы два файла:

```
SAMPLE.EXE     108752 байт
SAMPLE.IMA     394694 байт
Итого:                503446 байт
```

Заключение

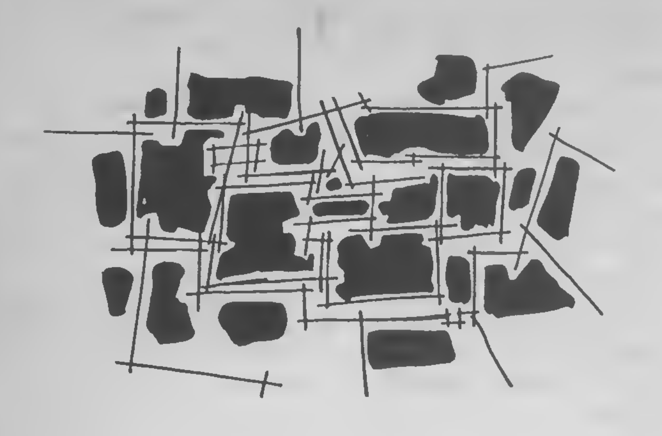
Итак, мы познакомились со средой Actor — объектно-ориентированным языком для создания Windows-про-

грамм. Нельзя сказать, что результат моего знакомства обрадовал меня: получить полмегабайта кода ради создания окна и меню, отображающего панель сообщения, — это, на мой взгляд, слишком дорогая цена. Да и способ создания приложения достаточно трудоемок (см. пп. 1-13). Загрузка такого приложения происходит тоже не очень быстро. Таким образом, мы можем смело вычеркнуть Actor из списка средств для создания Windows-приложений. Тогда, может быть, Actor можно использовать как средство для быстрого создания шаблонов приложений? И тут не выходит, так как после получения шаблона мы будем вынуждены каким-то образом "пристегивать" его к С- или Pascal-программе. Более того, для компиляторов С/С++ и Pascal существуют средства для создания шаблонов приложений (например, ProtoGen или Windows Maker). Библиотеки классов, входящие в состав Actor, уже реализованы для компиляторов С/С++ и Pascal, существуют соответствующие версии QELIB, для непрограммистов имеется Object Vision фирмы Borland, даже язык Basic пришел в среду Windows, принеся с собой технологию визуального программирования. Создатели языка Actor придумали неплохую концепцию полностью объектно-ориентированной среды для создания Windows-приложений, но, к сожалению, ее реализация очень далека от совершенства. Справедливость этого утверждения доказывается тем фактом, что на сегодняшний день не существует коммерческих Windows-приложений, созданных на языке Actor, кроме Whitewater Resource Toolkit фирмы Whitewater...

А. Федоров







Почему я взялся за это сочинение? Потому что мне нравится моя профессия программиста, нравится язык Модула-2. Я надеюсь поделиться этими скромными радостями с вами. Следует предупредить, что благодаря эмоциональной окраске известных положений этот текст гораздо ближе к изящной словесности (belle lettre), нежели к технической или научной статье.

Почему я работаю на языке Модула-2

Самое главное качество этого языка — простота[1]. Есть, конечно, и другие причины моего пристрастия к Модуле-2, но эта — основная. Кстати, простоту особенно ценит Н.Вирт — разработчик языков Паскаль, Модула-2 и Оберон.

Почему язык должен быть простым

Считается, что решение поставленной перед программистом задачи складывается из двух этапов:
- композиции (выбор методов решения) и
- кодирования (реализация выбранных методов).

Качество программы определяется в основном композицией (макроуровнем): дефекты композиции исправить намного труднее, нежели улучшить неэффективно написанный код (микроуровень). Если даже компилятор вырабатывает весьма эффективный код, то это еще отнюдь не залог эффективности вашей программы.

Но во многих случаях главное даже не эффективность — этап композиции определяет трудоемкость разработки (и дальнейшего сопровождения) программы. Хорошая композиция резко уменьшает время изготовления программы. Плохая композиция зачастую приводит к тому, что программа не будет закончена никогда — при затянутых сроках разработки заказчик теряет к ней интерес, появляются конкурентные продукты.

[1] Согласно моему личному опыту и мнению. Это примечание в последующем тексте не повторяется исключительно ради экономии бумаги.

Чаще всего быстро разработанная программа работает эффективнее, нежели аналогичная, но на разработку которой ушло больше времени. В этом нет ничего неожиданного. Если композиция лучше, то это сказывается одновременно и на сроках разработки, и на эффективности.

Так что же такое "хорошая" композиция? Ответ на этот вопрос лежит в области эстетики. Вообще, чем программист квалифицированней, тем сильнее эстетическое начало (конечно, это еще не значит, что хороший программист — всегда авторитет в искусстве составления букетов).

Простота языка означает прежде всего легкость применения эстетических оценок. Конечно, исследовав любой текст (например, на ассемблере или Си), можно восхититься его красотой или, наоборот, осудить его за корявость. Однако ассемблерный код чаще всего оставляет читателя равнодушным. Дело в том, что язык ассемблера очень сложен, и, для того чтобы оценить такой текст, нужно затратить значительные усилия. А чаще всего обходятся вовсе без них — заранее же не известно, стоит ли овчинка выделки, возможно, что никакой пользы извлечь не удастся. А если удовольствие в конце пути все же светит, то, вероятно, будет сильно подпорчено усталостью от усилий, связанных с его достижением.

Модула-2 — простой язык. Легко оценить качество программы, написанной на этом языке. Важно то, что оценить качество композиции ничуть не сложнее, чем качество кодирования.

А зачем, собственно, оценивать программу с точки зрения эстетики? Грубо говоря, программа либо рабо-

2. Зак. № 3781

тает, либо нет. Конечно, приятнее, если она выполнена элегантно, но конечный результат важнее (коньяк приятнее самогона, но "кайф" ловят и от того и от другого, и еще неизвестно, от чего быстрее). Заказчик платит за работу, а не за элегантность, ему не доступную и не нужную.

При композиции программы (важнейшем этапе разработки) единственный критерий окончания этого этапа — эстетический. Если у программиста эстетическое чутье не развито, он хватается за один из первых правдоподобных вариантов композиции; это пагубно сказывается и на сроках разработки, и на эффективности. Подсознательное предвосхищение гармонии — главный вожатый опытного программиста. Он начнет кодирование лишь тогда, когда проект станет удовлетворять его взыскательному вкусу. И результат будет разительно отличаться от результатов работы первого программиста, не чувствующего нюансов.

Развивать эстетический вкус можно двумя способами: анализируя чужие (в том числе бесспорные шедевры) или лишь свои собственные творения. Оба способа хороши, но первый легче. Однако программист чаще всего — как тот чукча, который не читатель, а писатель. Бесспорных шедевров мало, и изложены они чаще всего на малознакомых языках (см. например, Э.Дейкстра, "Дисциплина программирования" — М: Мир, 1978 г., 276 с.). Личный опыт бесценен еще по одной причине: по учебнику легко научиться кодированию, но почти невозможно научиться композиции. Искусство кодирования демонстрируется на маленьких фрагментах. Композиция предполагает наличие крупных задач, не вмещающихся в учебник. Учиться композиции можно только на практике (своей или своего коллектива). Известны отдельные крупные задачи, хорошо описанные в литературе (например, разработка компиляторов), но эти исключения погоду не делают — до всего приходится доходить своим умом.

Так вот: если программист заинтересован в повышении квалификации, то он вынужден развивать именно эстетическое чутье. Проведена композиция, выполнено кодирование, настает время оценить правильность выбранных решений. (Скорректировать косноязычные места, перекодировать снова, затем вновь произвести оценку...)



Именно в этом и заключен основной интерес профессии программиста — совершенству предела нет, и впереди новое состязание с энтропийным Хаосом. Это состязание выигрывается лишь в том случае, если Хаосу противопоставляется Гармония.

Модула-2 — простой язык, и именно его простота позволяет быстро совершенствовать эстетическое чутье. Работа на этом языке связана с удовольствием непрерывного профессионального совершенствования — сегодня я умнее, чем вчера, а завтра поумнею еще. Это — основная причина, по которой я работаю на Модуле-2.

Модули

Важнейшая техническая особенность Модулы-2 — этот язык специально разработан для создания крупных программ (программ, композиция которых как минимум не тривиальна). К сожалению, учебники ориентированы на крохотные задачи, в которых Модула-2 не имеет крупных преимуществ по сравнению, например, с Си.

Давайте рассмотрим концепцию модулей — основную концепцию языка Модула-2 (кстати, очень похожая концепция пакетов введена в язык Ада, также специально ориентированный на создание крупных программных комплексов).

Каждый логический модуль (за исключением главной программы и локальных модулей) хранится в двух автономных файлах (обычно с одним именем, но разными расширениями). Один из этих файлов называется модулем определений — он отвечает на вопрос, ЧТО делает этот модуль. Парный ему файл называется модулем реализации и отвечает на вопрос, КАК модуль устроен.

В модуль определения входят только описания — константы, типы, заголовки процедур — все, что может использоваться в других модулях. Обычно модуль определения содержит развернутые комментарии — инструкции по использованию этого модуля.

Модуль реализации включает все, что должно быть скрыто от внешних пользователей этого модуля, и в частности тела процедур, заголовки которых приведены в соответствующем модуле определения.

Другой модуль может использовать объекты, описанные в нашем модуле определения (но не в модуле реализации). Для этого другой модуль должен импортировать соответствующие идентификаторы — как минимум идентификатор (название) нашего модуля. Такие фразы импорта могут встречаться как в модулях определений, так и в модулях реализации.

Полезность фраз импорта удобно пояснить примером. Практически в каждой реализации Модулы-2 есть модуль Storage, ответственный за динамическое распределение оперативной памяти (он реализует кучу,

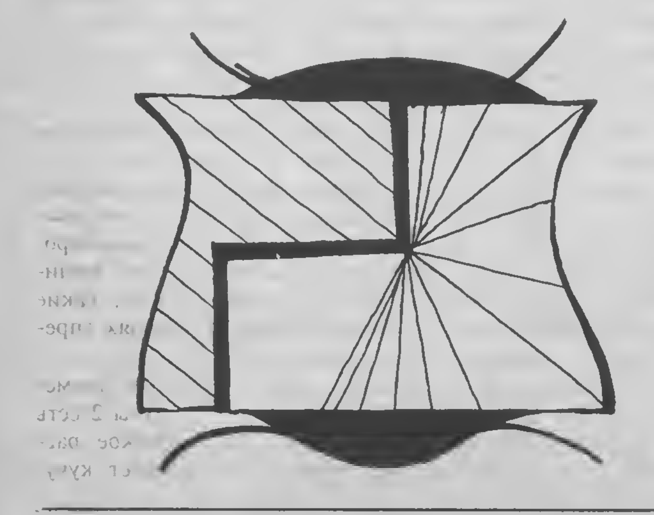
забирая из нее память процедурой ALLOCATE и возвращая ее процедурой DEALLOCATE). Пусть я работаю на ПЭВМ класса IBM PC и подключаю к ней добавочную память класса EMS. Чтобы уже написанная программа могла задействовать эту память, необходимо просто во всех фразах импорта заменить идентификатор Storage на идентификатор EMS (предполагается, что модуль EMS также экспортирует процедуры ALLOCATE и DEALLOCATE, которые позволяют реализовать кучу в добавочной памяти).

Итак, мы видим два принципиальных отличия от более ранних языков (например, Фортрана или Си):
- четкое разделение между определениями и соответствующей реализацией;
- обязательность фраз импорта.

Давайте поразмыслим, к каким последствиям приводят эти две особенности.

На этапе композиции составляются только модули определений (и главная программа). Взаимосогласованность модулей определений проверяется компилятором. Когда структура модулей "затвердеет", начинается этап кодирования — этап разработки соответствующих модулей определения. Таким образом, двум разным этапам разработки программы соответствуют различные объекты — единицы компиляции.

Если исправлен и перекомпилирован любой модуль реализации, то заведомо известно, что никакой другой модуль перекомпиляции не требует — весь импорт берется лишь из модулей определений. Однако если изменился какой-то модуль определений, то может последовать каскад других перекомпиляций — требуется перекомпилировать все те модули, которые импортировали объекты из измененного модуля (это будут модули реализации и, возможно, модули определений). Благодаря наличию фраз импорта очень легко собрать всю необходимую информацию о межмодульных зависимостях автоматически, и так же автоматически система может произвести каскад требуемых перекомпиляций. Другими словами, информацию о межмодульных связях (необходимую для поддержания целостности всей программы) предоставляет не программист, она собирается автоматически при помощи фраз импорта.



Все эти свойства приводят к тому, что программирующий на Модуле-2 и программирующий на Фортране или Си мыслят совершенно по-разному. Оба этих языка поддерживают концепцию независимой компиляции — программа состоит из отдельных модулей, причем за межмодульный интерфейс несет ответственность только сам программист — система почти ничем не помогает ему в поддержке этого интерфейса.

Характерная особенность мышления программиста на Си — боязнь серьезно затронуть исторически сложившийся межмодульный интерфейс. Конечно, существенную помощь оказывает программа make, но данные для нее нужно предоставлять вручную, и беда, если программист что-либо забыл или ошибся. Он интуитивно отталкивает мысль о серьезной ревизии композиции программы — обычно количество модулей относительно невелико, зато среди них есть и весьма крупные.

В противоположность этому, Модула-2 поддерживает концепцию раздельной компиляции — логическую непротиворечивость гарантирует сама разумно устроенная система программирования. Модули компилируются (и собираются) не независимо друг от друга, а на основе анализа времени и даты их создания (вспомним, что два файла, соответствующие одному логическому модулю, обычно имеют разные времена создания).

В условиях когда система гарантирует такую же целостность модульной программы, как если бы она была монолитной, психология разработчика меняется кардинально. Программист уже не озабочен поддержкой межмодульного интерфейса, он целиком сосредоточен на логической композиции модулей. Если выясняется, что разумно завести новый модуль, то он заводится немедленно — перекраивать граф межмодульных зависимостей вручную не придется, этот граф строится системой динамически и автоматически.

Любопытно, что концепция раздельной компиляции Модулы-2 заимствована в реализации Паскаля фирмой Borland International. Дело в том, что Паскаль изначально не имел средств ни независимой, ни раздельной компиляции — на нем можно было писать лишь монолитные программы (разумеется, речь идет о "чистом" Паскале). В силу этого расширить Паскаль в нужную сторону было легко. Однако концепция раздельной компиляции в Паскале фирмы Borland грубее, чем эквивалентная концепция Модулы-2 — в Паскале нет разделения логического модуля на два файла (определение и реализация).

Абстрактные типы данных

Вторая важная особенность Модулы-2, связанная с разработкой больших программ, — это поддержка абстрактных типов данных. Вероятно, не очень удобное

название — почему абстрактные типы, если моя задача вполне конкретна.

Абстрактный тип данных — это неразрывное целое некоторого типа (детали внутреннего строения которого не важны) и совокупность процедур, реализующих все операции над этим типом (в последние годы такие процедуры почему-то стали часто называть по-особому — "методами"). Прилагательное "абстрактный" употребляется не в смысле "бесполезный на практике", а означает лишь то, что внутреннее устройство такого типа с точки зрения пользователя совершенно не важно.

Пусть в некотором модуле определений описаны какой-то тип (приведен лишь его идентификатор, все подробности отсутствуют, они находятся в соответствующем модуле реализации) и заголовки процедур, некоторые параметры которых принадлежат этому типу (тела этих процедур также располагаются в модуле реализации). Другой модуль может импортировать тип и процедуры, заводить переменные этого типа (например, массивы или списки объектов этого типа), но работать с переменными этого типа можно при помощи только соответствующих процедур.

В чем преимущество такого подхода? В надежности. Абстрактный тип характеризуется лишь идентификатором. Если я по ошибке попытаюсь обработать переменную абстрактного типа "чужой" процедурой, то компилятор обязательно заметит эту ошибку.

Взаимосвязь объектов, описанных в одном модуле определений, может быть разной. Степень этой взаимосвязи определяет информационную "прочность" модуля.

Модуль может экспортировать совокупность процедур, которые обычно применяются вместе, — это единственное, что их связывает. Например, таким может быть модуль работы со строками, длина которых задана статически (то есть известна программисту). Такой модуль включает, в частности, процедуры для вставки, сравнения, конкатенации строк.

Более прочными выглядят модули, которые экспортируют совокупность процедур, работающих с общими данными. В модуле определений сами эти данные не фигурируют — они скрыты в модуле реализации. Примером такого модуля служит модуль Storage — куча скрыта в модуле реализации, а модуль определения содержит заголовки процедур, которые совместно работают с этой кучей.

Самые же прочные модули те, что реализуют абстрактный тип данных. Такой модуль может, например, реализовывать файловую систему: он экспортирует идентификатор файлового типа и типичные операции над файлами — создание, открытие, чтение, запись и пр. Преимущества абстрактных типов широко распропагандированы авторами большинства реклам объектно-ориентированного подхода к программированию, так что скорее всего они вам известны.

Заметим, что степень прочности модуля зависит от объема скрываемой им информации. Модуль первого класса скрывает только тела процедур, модуль второго класса скрывает еще и переменные, общие для нескольких процедур (возможно, и типы, соответствующие этим переменным). Значение этих переменных между вызовами соответствующих процедур сохраняются. Наконец, модуль третьего класса (реализующий абстрактный тип данных) скрывает больше всего — помимо тел процедур один или даже несколько типов, идентификаторы которых приведены в модуле определений.

Эта классификация сильно помогает при оценке проекта композиции. Чем больше в проекте модулей третьего класса (абстрактных типов данных), тем проект элегантнее. И наоборот, проект, состоящий преимущественно из модулей первого класса, почти наверняка плох, композиция его произведена "без души". Такой проект скорее всего будет неэффективен и дорог в реализации.

Менее важные концепции

Важная особенность Модулы-2 — возможность полного доступа к аппаратуре ЭВМ. Собственно говоря, язык задумывался так, чтобы на нем можно было писать все, что можно написать на ассемблере (и примерно с той же степенью эффективности).

Большинство горячих приверженцев языка Си даже не подозревают, что на нем можно написать не любую программу, а любую последовательную программу. Язык Си (как и Паскаль) не поддерживает концепции процессов (участков программы, выполняющихся параллельно или квазипараллельно). Концепция процессов — ключевой элемент так называемых программ реального времени, управляющих периферийным оборудованием (или хотя бы считывающих с них информацию). Разумеется, в рамках операционной системы реального времени такие программы можно писать на любом языке (в том числе и на Си), но работа с процессами (и межпроцессной взаимосвязью) в этом случае производится не в терминах языка программирования, а при помощи средств операционной системы — как правило, низкоуровневых и от языка программирования не зависящих. Отметим кстати, что MS-DOS (как и все надстройки над ней — например, Windows) не является операционной системой реального времени.

Язык Модула-2 содержит концепцию сопрограмм — старого и относительно малоизвестного аппарата, являющегося расширением общепринятого аппарата процедур (см. Marlin Ch.D., Coroutines. — Berlin, Springer, 1980, XII+256 p.; рецензия на эту книгу опубликована в сборнике "Новые книги за рубежом", серия А, 1981, № 9, с. 29-32). Низкоуровневый аппарат сопрограмм можно использовать для моделирования квазипараллельных процессов, выполняющихся "одновременно" на единственном процессоре (даже в рамках однозадачной системы, такой как MS-DOS).

В разных задачах реального времени предъявляются различные требования к аппарату процессов. Этот ап-

парат как таковой в Модулу-2 не встроен, но при помощи сопрограмм можно написать модуль, реализующий абстрактный тип процесса и наиболее полно учитывающий специфику данной задачи реального времени (обычно такой модуль называется "диспетчером").

Аппараты сопрограмм и процессов полезны не только в программах реального времени, они мощные (хотя и нетрадиционные) средства структуризации.

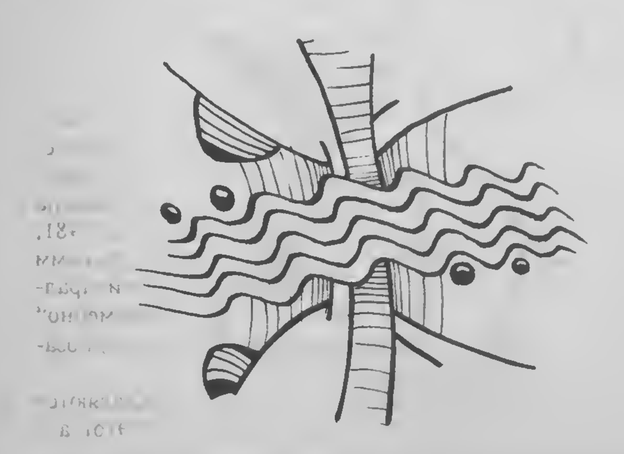
Например, известная учебная задача заключается в выравнивании текста, состоящего из слов, по левой и правой границам. Если язык допускает использование сопрограмм, то эта задача становится тривиальной.

Язык Модула-2 не содержит аппарата исключительных ситуаций (exceptions), облегчающего создание робастных (особо надежных) программ. Для моделирования этого аппарата удобно использовать процессы — по одному на каждый тип исключительных ситуаций. Каждый такой процесс практически не требует накладных расходов: он устанавливает в стеке обработчики исключительных ситуаций (Модула-2 позволяет работать с процедурными типами) и ожидает появления исключительной ситуации. При ее появлении вызывается последний установленный обработчик, после чего управление возвращается в ту точку, где ситуация возникла.

Наконец, процессы удобны при моделировании тех задач, в которых процессы существуют изначально. Например, пусть имеется задача моделирования волейбольной команды. Традиционное решение заключается в том, чтобы написать процедуру, моделирующую игрока, а также некий монитор, поочередно вызывающий эту процедуру — управляющий всеми игроками и координирующий их поведение.

Однако в действительности никакого объекта, стоящего над игроками, нет. Традиционная модель сильно отклоняется от моделируемой реальности.

Процессы позволяют упростить модель. Напишем процесс, соответствующий игроку, и запустим шесть экземпляров этого процесса. Каждый игрок будет самостоятельно взаимодействовать с другими пятью игроками команды и с мячом. Такая модель ближе к моделируемой реальности, поэтому она, безусловно, проще.



Поверхностное знакомство с языком Модула-2 может создать впечатление, что в языке отсутствуют встроенные средства работы с битами. Это впечатление ошибочно. Просто работа с битами в Модуле-2 высокоуровневая, она опирается на фундаментальную математическую концепцию множества (а реализуются множества в Модуле-2 обычно при помощи машинных операций, работающих с битами).

Вообще говоря, Модула-2 — язык со строгой типизацией (при которой в исходный текст вносится дополнительная избыточность, служащая для выполнения автоматического добавочного контроля). Однако при помощи явных средств самого языка строгий контроль типов можно обойти.

Чего мне не хватает в Модуле-2

Условно можно считать, что главное достижение программирования 60-х годов — структурное программирование, 70-х — абстрактные типы данных, 80-х — объектно-ориентированное программирование. Модула-2 разработана около 1979 года, и объектно-ориентированных средств этот язык, к сожалению, не содержит. Объектно-ориентированные средства — это языковая концепция, облегчающая разработку больших программных проектов (добавочная по отношению к раздельной компиляции и абстрактным типам данных). Давайте с помощью примера разберем те трудности, с которыми я сталкиваюсь приблизительно в 5% разрабатываемых мною модулей (в остальных 95% я успешно борюсь со сложностью проекта при помощи абстрактных типов данных).

Пусть некий модуль содержит абстрактный тип данных довольно сложной структуры (разумеется, структура такого типа расшифровывается в модуле реализации). Как правило, такой тип связан с вариантной записью — полями, общими для всех случаев, дискриминантом, который задает, какие добавочные поля фактически присутствуют, и самими добавочными полями, номенклатура которых зависит от текущего значения дискриминанта (для программистов на Си: вариантная запись).

Трудности, связанные с реализацией такого абстрактного типа, чисто количественные (никаких логических трудностей нет). Сложный тип данных, как правило, требует крупных процедур, которые, конечно, можно разбить на более мелкие, но все они должны содержаться в одном модуле реализации, объем которого сильно вырастает (иногда — до нескольких тысяч строк). Однако разбить такой модуль на несколько невозможно — в языке Модула-2 можно либо скрыть реализацию типа целиком (абстрактный тип), либо вынести все детали строения типа в модуль определений (программист, привыкший к удобствам и элегантности абстрактного типа данных, обычно идет на это крайне неохотно). Никаких промежуточных вариантов нет — нужно либо все скрыть (модуль реализации сильно разрастается, и работать с ним становится трудно), ли-

бо все открыть (тогда обработку этого типа можно разнести по нескольким модулям, импортирующим сложный тип, но теряется защита).

Модула-2 содержит аппарат локальных модулей. Разумеется, в модуль реализации я вставляю локальные модули, которые несут ответственность за обработку общих полей и каждого набора вариантов добавочных полей. Локальные модули упрощают структуру модуля реализации, но его объем не уменьшают.

Если бы язык Модула-2 содержал аппарат наследования, являющийся ключевым в объектно-ориентированном программировании[2], то я решил бы эту задачу следующим образом.

Сначала я создал бы модуль, реализующий абстрактный тип данных — поля сложной записи, общие для всех вариантов.

Затем написал бы несколько (n) производных модулей, базирующихся на первом модуле. Каждый такой производный модуль соответствовал бы одному варианту добавочных полей, и, возможно, переопределял некоторые методы базисного модуля.

Наконец, я создал бы сводный модуль, файл определений которого полностью совпадал с "настоящим" файлом определений, — этот модуль экспортирует абстрактный тип, соответствующий сложной записи. Однако соответствующий модуль реализации был бы весьма прост: он импортировал бы 1+n имеющихся модулей. При вызове каждой экспортируемой процедуры проверялся бы дискриминант (при помощи некоторого метода) и вызывался бы нужный метод, соответствующий текущему варианту добавочных полей.

Если бы Модула-2 допускала наследование (и, как следствие, полиморфизм), то мне бы удавалось разбивать по-настоящему большие модули на ряд мелких. Однако, к счастью, большие модули встречаются редко. Кроме того, разумно спроектированная оболочка (в частности, хороший редактор исходных текстов) позволяет без особых хлопот справляться и с файлами, состоящими из нескольких тысяч строк.

Таким образом, я чувствую, что объектно-ориентированное программирование — вещь хорошая, и сожалею, что она мне недоступна.

Как уже говорилось, Модула-2 позволяет работать с переменными процедурных типов. Это означает, что таблицы методов для классов можно строить в Модуле-2 "вручную". Однако из-за отсутствия специальной языковой поддержки это — довольно утомительное (и ненадежное) занятие.

Почему же я не перехожу на C++, который содержит аппарат и абстрактных типов данных, и наследования? Мне представляется, что, к сожалению, эти две важных и сложных программных концепции в C++ логически не разделены. В результате этот язык чрезвычайно сложен (Модула-2 значительно проще языка Си, что уж тут сравнивать ее с C++), и мне работать на нем было бы трудно (особенно первые 6-8 лет). Кроме того, в C++ отсутствуют многие приятные "фенечки", к которым я привык (вроде сопрограмм, разделения логического модуля на два файла) и которые кажутся мне полезными.

Заключение

Модула-2 — не идеальный, но вполне достойный язык общего назначения. Он особенно удобен, если требуется разработать нетривиальную программу, которая должна работать эффективно.

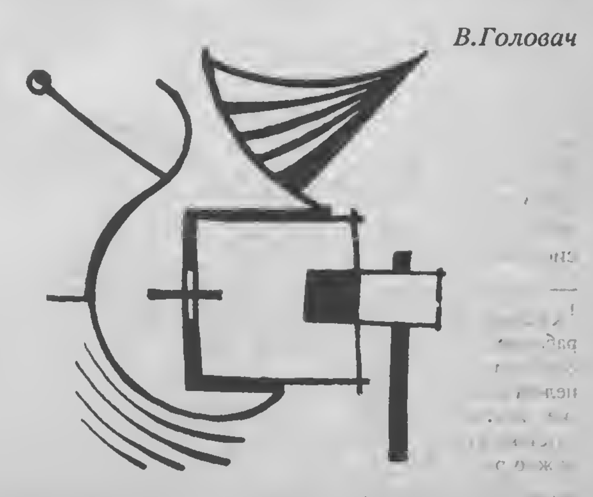
К сожалению, международный стандарт на Модулу-2 пока отсутствует. Несмотря на это, язык коммерчески реализован практически на всех платформах и, возможно, со временем вытеснит Паскаль.

Если бы я работал не в MS-DOS, то, вероятно, предпочел бы какой-либо другой язык — Ада, Модула-3 или Оберон. Однако реализации этих языков слишком тяжелы для компьютеров класса IBM PC. Мне кажется, что в рамках MS-DOS коммерческих реализаций этих языков не будет никогда.

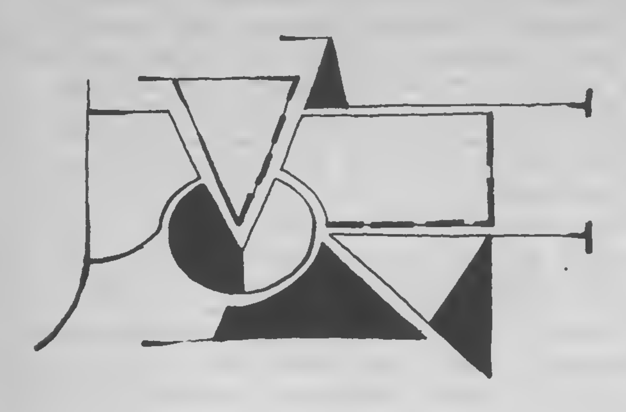
Что касается Ады (торговый знак Министерства обороны США) и Модулы-3 (совместная разработка фирм Digital Equipment Corporation и Olivetti), то это весьма сложные языки, потенциально способные вырабатывать эффективный (по времени и по памяти) код. Язык Оберон поддерживает объектно-ориентированное программирование и намного проще Модулы-2 (Н.Вирт считает, что Оберон ближе к Модуле-2, нежели Модула-2 — к Паскалю). Однако похоже, его реализации используют оперативную память весьма расточительно, и скромные возможности адресации MS-DOS — серьезное препятствие для возможного реализатора.

Разумеется, при выборе языка для практического решения задач достоинств языка как такового мало — необходимо оценить надежность и удобство реализации, наличие нужных библиотек. Но эта важная тема выходит за рамки данной статьи.

В.Головач



[2]Некоторые реализации Модулы-2 (например, Top Speed фирмы Clarion) объектно-ориентированны. Однако реализация Top Speed мне не нравится — в ней много ошибок, плохой отладчик, неудобная оболочка. Я работаю на системе Logitech Modula-2, в которой наследования нет (пока?).



Замечания по системе программирования Logitech Modula-2

Введение

С Logitech Modula-2 я работаю с конца 1988 года. Сначала я работал на версиях 3.00 и 3.03. Однако в начале этого года я перешел на более современную версию — 4.00. Она появилась в продаже на Западе в первой половине 1992 года, а в Москве стала продаваться в декабре того же года.

В этой заметке я пытаюсь суммировать свой опыт работы с этой системой программирования. Разумеется, основное внимание уделю именно последней версии.

Мягко говоря, Logitech Modula-2 в нашей стране была непопулярна — основная масса программирующих на Модуле работала (и работает) на системе Top Speed фирмы Clarion (ранее — JPI). Система Top Speed имела три важных преимущества перед Logitech Modula-2 версии 3 — она продавалась в СССР за рубли, создавала более эффективный объектный код и имела оболочку, похожую на привычные оболочки фирмы Borland. Кроме того, Top Speed намного дешевле (в долларах) системы фирмы Logitech[1]).

Третья версия Модулы-2 фирмы Logitech фактически не имела оболочки (то есть программы, которая объединяла все нужные программисту инструменты). Роль оболочки играл универсальный редактор текстов программ Point, снабженный некоторыми примочками, облегчающими работу с Модулой (из редактора можно было вызывать некоторые программы системы программирования — например, компилятор, компоновщик). Этот редактор будет обсуждаться ниже, однако здесь следует сказать, что именно отсутствие оболочки, похоже, подтолкнуло отечественных программистов к системе Top Speed.

[1]Казалось бы, в условиях, когда большинство программистов работают с пиратскими копиями, это роли не играет. Однако согласно моим наблюдениям, дешевые программные продукты нелегально копируются значительно охотнее, чем более дорогие (и стало быть, редкие). Возможно, причина этому — отсутствие рынка программного обеспечения: крадется не то, что нужно реально, а то, что есть у приятелей.

Основное впечатление от версии 3 фирмы Logitech — исключительная надежность. За годы работы с компилятором и отладчиком я не столкнулся практически ни с какими ошибками. Кому как, а мне нравится работать с надежной системой. Мой недолгий опыт работы с системой Top Speed показал, что надежность — это не органическое свойство реализаций Модулы на IBM PC, и я быстренько вернулся к реализации Logitech.

Так как я работал на системе Logitech постоянно, то примириться с отсутствием оболочки я не мог. Для третьей версии мной была написана оболочка, интегрирующая все нужные программы. По моему мнению, она значительно превосходила то, что предлагала (и предлагает) своим пользователям фирма Borland.

Пакет Logitech Modula-2 version 4.0

Основные отличия версии 4 от предыдущих версий таковы:
- сильно упрощены и улучшены возможности стыковки с другими системами программирования;
- увеличена эффективность объектного кода (в частности, за счет введения моделей памяти), и
- допускается программирование не только в среде MS-DOS, но и в среде MS Windows.

Теперь перейдем к более подробному обзору пакета Logitech Modula-2 v. 4.0.

Упаковка. На упаковке указаны название пакета — Logitech Modula-2 v. 4.0 и разработчик — фирма Multiscope. Эта фирма выделилась несколько лет назад из фирмы Logitech, выкупив права на систему программирования и на известное (на Западе) название.

На коробке приводится также таблица, сравнивающая эффективность объектного кода Модулы-2 и компиляторов Borland C++ v. 3.0 и Microsoft C v. 6.0. Из анализа таблицы можно сделать вывод, что эффектив-

ность Logitech Modula-2 v. 4.0 несколько выше. Утверждается также, что компилятор Модулы-2 работает в 3 раза, а компоновщик — в 20 раз быстрее, чем компилятор и компоновщик С фирмы Microsoft.

Компилятор. Компилятор написан по лицензии фирмы Stony Brook Software. В отличие от реализации Top Speed, компиляции подлежат также модули определения.

Язык несколько расширен относительно стандарта Н.Вирта: часть расширений связана с введением моделей памяти и возможностью стыковки с другими языками, но также введены возможности записи структурных констант и использования в идентификаторах знака подчеркивания. Несколько ослаблена совместимость встроенных числовых типов разной длины.

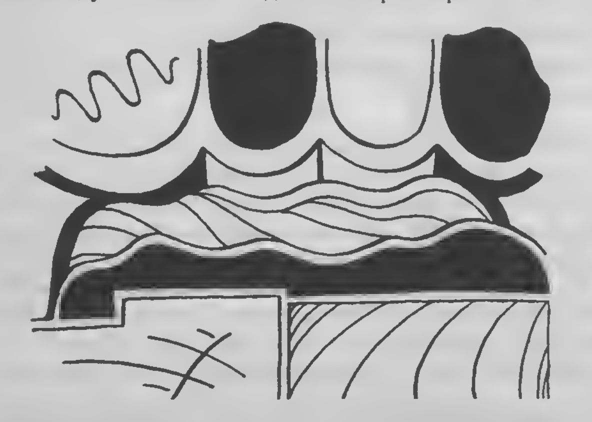
Компилятор воспринимает многочисленные ключи и псевдокомментарии, служащие в основном для оптимизации генерируемого кода и вызова процедур, написанных на других языках. Есть возможность условной компиляции (польза от которой довольно сомнительна — см. ниже).

Обычно компилятор генерирует файл OBJ, но может генерировать и файл LIB. Это дает компоновщику возможность включать в программу не весь код модуля, а код лишь фактически требуемых процедур (smart linking).

Компоновщик. На самом деле система содержит два компоновщика — для MS-DOS и MS Windows. Оба сделаны по лицензии фирмы SLR Systems, управляются схожим образом, и ниже речь идет сразу о двух компоновщиках.

Компоновщик универсален в том смысле, что позволяет обрабатывать любые файлы OBJ и LIB. Благодаря этому на Модуле-2 может быть написана лишь часть программного проекта (другие части можно писать на других языках). Кроме того, он поддерживает "прозрачные" оверлеи: для их создания модифицировать исходный код не требуется (например, в исходный код не нужно импортировать специальный модуль управления оверлеями, как в версии 3).

Однако в одном важном отношении по сравнению с версией 3 компоновщик ухудшился. Дело в том, что язык Модула-2 может и должен гарантировать логическую целостность программы, составленной из раздельно скомпилированных модулей. Для этого система должна производить контроль версий. Частично такой контроль производится компилятором, однако полный контроль на этапе компиляции невозможен. В версии 3 компоновщик завершал этот контроль и делал все то, что не мог делать компилятор. В версии 4 свою часть работы компоновщик не выполняет. Это означает, что некоторые ошибки программиста, связанные с межмодульным интерфейсом, система может не заметить. Другими словами, в этой версии открыта лазейка суровым ошибкам, типичным для языка Си. Разумеется, программисту, давно работающему на Модуле и уже отвыкшему от ошибок такого класса, смириться с этим недостатком весьма трудно.



Служебные программы. В пакет входит несколько служебных программ.

Новой является лишь одна — M2IMPORT. Модула-2 требует явной спецификации всех объектов из окружения — списков импорта. Эта программа позволяет писать модуль, не задумываясь о списках импорта. После того как модуль написан, M2IMPORT просматривает текст модуля и формирует списки импорта автоматически; в случае конфликтов имен она обращается за помощью к программисту.

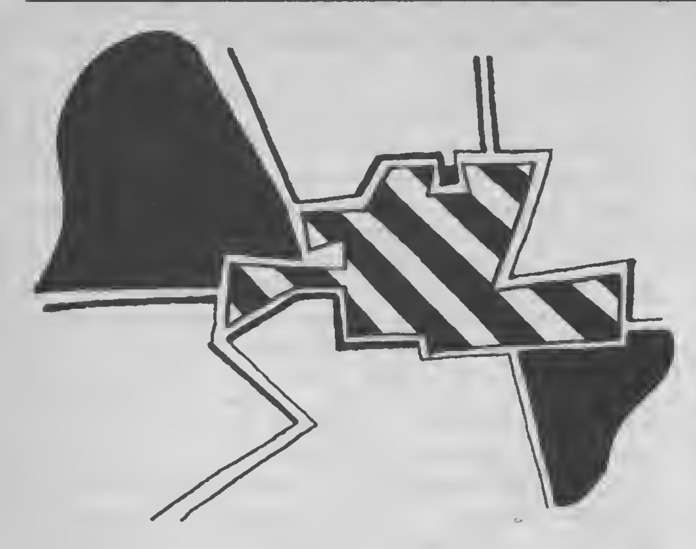
M2CHECK детально исследует исходный текст модуля и сообщает о "подозрительных" местах — возможных семантических ошибках (например, описанных, но не задействованных идентификаторах). Это — полезная программа, но неудачно то, что компилятор тоже может выдавать предупреждения. Разумнее было бы сосредоточить все предупреждения в одной программе.

M2VERS позволяет писать единый исходный код для нескольких версий модуля (например, рабочий код модуля и сокращенный код его демонстрационной версии). Эта программа просматривает текст и выделяет код, адекватный нужной версии. Это весьма полезное средство для написания сложных систем (более удобное, нежели универсальный макропроцессор), и благодаря ему средства условной компиляции компилятора фактически избыточны.

M2CONVERT частично автоматизирует перевод программ с версии 3 на версию 4.

Безусловно, самая важная служебная программа — M2MAKE. Она анализирует списки импорта и времена создания модулей, выясняет необходимость перекомпиляции некоторых исходных файлов и генерирует командный файл, содержащий вызовы компилятора и компоновщика. После выполнения этого командного файла гарантируется логическая целостность программы, состоящей из ряда модулей. При использовании этой программы у меня возникли трудности (см. ниже).

Две старые служебные программы исключены. Программа генерации ассемблерного листинга стала ненужной (программу на ассемблере может теперь выдавать компилятор). С потерей программы форматирования примириться труднее.



Программа M2FORMAT в версии 3 позволяла элегантно оформлять синтаксически корректный исходный текст для выдачи на экран или на принтер. Форматирование текста осуществлялось так: программист оформлял шаблон требуемым образом, а при запуске эта программа форматировала указанный модуль так, как был оформлен шаблон. Благодаря этому все модули можно было оформлять строго единообразно, что чрезвычайно повышало удобство их чтения.

Полезность такого оформления трудно объяснить тому, кто им никогда не пользовался. Это все равно что рассказывать о преимуществах использования мыши человеку, который работал только с клавиатурой: пока не начнешь работать, удобства остаются умозрительными, но когда привыкнешь — отказаться от них уже очень трудно. Исключение служебной программы M2FORMAT из нового пакета было для меня тяжелым ударом.

Библиотека. В основном библиотека предыдущей версии сохранилась. Главное отличие — несколько изменен модуль SYSTEM, который содержит аппаратно-зависимые средства. Безусловно, новая версия этого модуля стала гибче и строже, однако появились определенные трудности, связанные с преобразованием ранее написанных модулей.

Добавились модули, связанные с программированием в MS Windows. Выкинут модуль текстовой оконной системы. (Эта оконная система у меня энтузиазма не вызывала. Зато теперь можно использовать практически любую оконную систему, написанную, например, на Си.)

Исходные тексты имеются для всех модулей системной библиотеки.

Интересная особенность моделей памяти — модуль, скомпилированный в одной из малых моделей, может вызывать процедуры, написанные в максимальной модели. В этой связи имеется лишь один экземпляр системной библиотеки (точнее, два — для DOS и Windows), скомпилированный в максимальной модели памяти. Если очень хочется увеличить эффективность небольшой программы, то можно самому перекомпилировать в нужном режиме исходные модули библиотеки.

Модули, обеспечивающие прерывания DOS и выполнение функций Windows, комментариев не содержат вообще. Предполагается, что программист имеет какую-то иную документацию по этим системам.

Отладчик. На самом деле система содержит несколько отладчиков — для MS-DOS и MS Windows, для маломощных процессоров и процессоров начиная с 386, для автономной и удаленной (по сети Novell или по связи RS-232) отладки. Отладчики для MS-DOS работают в текстовом режиме, а для MS Windows — в графическом (с добавочными возможностями).

Помимо этих традиционных динамических отладчиков, имеются также два "посмертных" отладчика (для DOS и Windows). Если прикладная программа импортирует соответствующий модуль, но в ходе выполнения происходит семантическая ошибка (например, попытка разыменования пустого указателя), то информация о текущем состоянии программы (в частности, значения всех переменных) записывается в специальный файл, и после запуска "посмертного" отладчика можно проанализировать причины, приведшие к ошибке.

Естественно, пользовательский интерфейс в разных отладчиках одинаков, и ниже речь идет сразу обо всех.

Простотой отладчик напоминает прекрасный отладчик версии 3, но имеются серьезные функциональные расширения.

Самое главное — отладчик многоязыковой, то есть позволяющий отлаживать программу, написанную на разных языках (и на системах разных фирм). Например, выражения для отслеживаемых переменных можно вводить согласно разному синтаксису (Си, Бейсик, Фортран, Паскаль, Модула-2). Имеется даже возможность просмотра классов (которые имеются в C++, но отсутствуют в Модуле-2).

Этот отладчик — первый целиком самостоятельный продукт фирмы Multiscope, и он (безотносительно к Модуле-2) хорошо известен на Западе.

По входной информации отладчик Multiscope совместим со стандартным отладчиком CodeView фирмы Microsoft, так что исходные данные для него может поставлять фактически любой компилятор.

Когда отладчики Multiscope в Москве отдельно еще не продавались, минимум один пакет Модулы-2 был куплен только ради отладчика (покупатели работали на Си и переходить на Модулу-2 не собирались).

Оболочка. Оболочка для MS-DOS выполнена на базе купленного у фирмы Logitech универсального редактора Point. Если Point не допускал работы без мыши, то оболочка пакета позволяет эмулировать мышь с клавиатуры и содержит несколько средств, специфичных для Модулы-2. Запускать программы пакета (кроме отладчика) можно из оболочки, причем они выполняются в окнах. Можно проверять синтаксис исходного кода в окне и вводить заготовки (часто применяемые

конструкции языка). В целом оболочка имеет довольно традиционный вид.

Оболочка для MS Windows выполнена в виде диалоговой панели, которая содержит имена файлов и кнопки, соответствующие редактору, компилятору, отладчику и т.п.

Естественно, качество оболочки сильно зависит от редактора. Поэтому рассмотрим свойства редактора более подробно.

Это самый мощный и удобный редактор текстов программ из тех, которые я видел. Однако он категорически противопоказан тем, у кого нет мыши. Если у вас имеется мышь (лучше трехкнопочная, но весьма эффектно работает и двукнопочная), то с этим редактором поэкспериментировать стоит.

Удобство редактора целиком связано с богатством функций, навешиваемых на мышь. Например, для копирования фрагмента текста предусмотрено две разные функции, применяемые в различных ситуациях. Можно сначала отметить точку, куда следует скопировать, а затем выделить копируемый текст (КУДА, затем ЧТО). Можно сначала выделить текст, а затем указать, куда следует его скопировать (ЧТО, затем КУДА). Обе этих функции можно выполнять только с помощью мыши, не касаясь клавиатуры — это сильно ускоряет копирование. Там, где обычно целесообразно ввести идентификатор повторно, в редакторе Модулы-2 проще скопировать имеющийся идентификатор.

Следует также отметить удобство работы с окнами. Например, поиск можно проводить не в одном окне, а в нескольких, причем исключив некоторые из них из поиска (кстати, поиск выделенного фрагмента производится также одной мышью, без участия клавиатуры). Модули определений системной библиотеки проще вызывать в окна редактора, чем искать и просматривать их в печатном руководстве.

Редактор можно перестраивать. Любую функцию редактора можно связать с клавишей (или комбинацией клавиш), нажатием кнопки мыши (в зависимости от текущего местоположения указателя и даже от направления его движения) или альтернативой меню (меню и подменю также могут строиться пользователем).

Однако удобство выполнения функций затрудняет обучение — чтобы научиться использовать редактор эффективно, надо учиться. Для облегчения обучения поставляемая фирмой настройка приближена к возможностям среднеарифметического редактора. На мой взгляд, в этом заключается крупный просчет: достоинства редактора в значительной степени скрыты, и начинающий пользователь не хочет его изучать.

Тот же товар можно было бы подать гораздо эффектнее, если бы изначально редактор был настроен "круто" (и отдельно на мышь с двумя или тремя кнопками), но снабжен оперативным уроком (диалоговую программу изучать гораздо проще не по документации, а "на ходу"). Система оперативных консультаций редактора (help) связана с поставляемой настройкой, и при перенастройке редактора эти консультации становятся бесполезными (компилятора консультаций нет).

Документация. Документация состоит из четырех книг:
- руководство по оболочке (и ее конфигурации);
- описания языка и системной библиотеки;
- руководство по отладчикам;
- справочник программиста (описания всех программ, кроме отладчиков, особенности реализации, программирование в MS Windows).

Кроме того, имеются три текстовых файла — дополнения и исправления печатной информации.

Почти все написано хорошо или приемлемо, но материал, посвященный оболочкам MS Windows и M2MAKE, изложен плохо. Например, возможность работы с заготовками в оболочке не описана вообще. Построение динамических библиотек DLL практически не раскрыто — предлагается разобраться в приведенном примере. Документация по M2MAKE чрезвычайно лаконична. Вероятно поэтому эта служебная программа работает у меня странно — требуются заведомо лишние перекомпиляции (это не страшно, но впустую расходуется время).

Что можно сделать лучше

Этот краткий обзор намечает некоторые возможные пути улучшения пакета Logitech Modula-2 version 4.0. Однако вряд ли стоит ждать, пока фирма выпустит новую версию — кое-что можно сделать самостоятельно.

Как уже говорилось, я работал на Модуле версии 3 в рамках своей оболочки. Качество ее было (по моему мнению и по отзывам тех, кому я ее передал) высоким. Например, когда я экспериментировал с пакетом Top Speed, то был просто вынужден написать для него аналогичную оболочку (в частности, интегрировав туда любимый редактор Point) — без нее работать с этим пакетом мне было трудно.

Немедленно после покупки версии 4 я приступил к разработке аналогичной оболочки и для нового пакета. Фирменная оболочка не удовлетворила меня по следующим причинам:
- не гарантируется логическая целостность программы;
- необходимость индивидуального управления многочисленными ключами компилятора, компоновщика и т.п. приводит к тому, что требуется держать в памяти знания о всей системе программирования;
- при работе с многочисленными окнами легко ошибиться и послать на компиляцию совсем не тот файл, который нужно. Это может привести к тому, что волей-неволей придется проделать еще ряд корректирующих перекомпиляций (о странном поведении M2MAKE уже сказано выше);
- из оболочки для MS-DOS отладчик запустить нельзя;
- отсутствует ряд привычных и удобных возможностей, которые имелись в моей оболочке раньше.

Самодельная оболочка для пакета Logitech Modula-2 version 4.0 обладает многими преимуществами, о которых следует сказать подробно.

1) Оболочка выполняется под MS-DOS, но позволяет разрабатывать и отлаживать программы и под MS Windows (тем самым отпадает необходимость работы с двумя оболочками).

2) Она не погружена в редактор, а управляется при помощи меню (редактор интегрирован точно так же, как и почти все остальные программы). Если компилятор выявил ошибки, то происходит автоматический возврат в редактор, и перемещаясь по ошибкам, их можно корректировать.

3) Файлы проекта создаются и поддерживаются автоматически. Пользователь может ничего не знать про ключи программ. Для каждого нового проекта оболочка предъявляет пользователю диалоговую панель, поля которой требуется заполнить (или оставить значения по умолчанию). Эти поля, в частности, задают модель памяти и конечное назначение программы (DOS, Windows или DLL). В ходе работы над проектом выбранные ранее значения полей можно скорректировать.

4) Работая в этой оболочке, пользователь может быть уверен, что логическая целостность программы сохраняется. Когда возможно, компилятор вызывается напрямую. Если прямой вызов компилятора может нарушить целостность, то вызывается программа M2MAKE (в некоторых автоматически выявляемых ситуациях — из третьей версии), и число лишних перекомпиляций сводится к минимуму.

5) Если расширения языка не задействованы, то можно запустить программу форматера из версии 3. Модули определений обычно форматировать автоматически можно, но, к сожалению, модули реализации во многих случаях приходится оформлять вручную (это связано с расширениями языка). Буду весьма признателен тем, кто сможет поделиться со мной свободно распространяемой программой интеллектуального форматирования, настраиваемой на разные языки. Если такой программы ни у кого нет, то, возможно, придется писать такой форматер самому.

6) Предусмотрена возможность ведения архивов модулей. Исходный текст отлаженного модуля можно сохранить на дискете (оболочка проверяет правильность вставленной дискеты, так что риск ошибки мал). Это уменьшает объем текущего рабочего каталога на жестком диске и упрощает отладку (архивированный модуль отладчик не “замечает“). Каждый пользователь может иметь личный архив.

7) Предусмотрена возможность “отката“, когда после экспериментирования появляется желание вернуться к старому варианту. Число шагов отката не ограничено.

8) В языке Модула-2 переименование модуля — довольно хлопотное дело. Недостаточно переименовать и модифицировать один модуль, требуется также изменить тексты всех модулей, которые импортируют измененный. В оболочку включена служебная программа M2REN, изменяющая имя модуля.

Разумеется, не стоит приводить полное описание возможностей новой оболочки Модулы-2. Важно лишь отметить, что она очень проста в использовании и обеспечивает высокую надежность программирования.

Рассказ о моей оболочке содержит иллюстрацию одной важной посылки: если вас что-то не устраивает в применяемой вами системе программирования, не миритесь с неудобствами, а пытайтесь исправить их. Конечно, на это потребуется время, но для профессиональной работы (комфортабельных условий) затраченное время того стоит.

В заключение хочу поблагодарить Николая Семенова, принимавшего активное участие в разработке всех упомянутых оболочек, и Михаила Жукова, участвовавшего в разработке оболочки для Logitech Modula-2 v. 3.

В.Головач



Графический интерфейс для среды MS-DOS

Введение

В этой статье рассматриваются некоторые вопросы, касающиеся разработки Windows-подобного графического интерфейса для среды MS-DOS. В наших условиях такая разработка имеет некоторый смысл, поскольку сама система Microsoft Windows предъявляет достаточно жесткие требования к ПЭВМ.

Объем статьи ограничен, и мы рассмотрим лишь метод формирования графического экранного изображения, приведем данные о его быстродействии и выясним, в какой степени он обеспечивает аппаратную независимость прикладной программы. Излагаемый метод не претендует на универсальность, но он может быть полезен при разработке приложений, включающих окна с графиками функций, различными диаграммами и т.п.

Все приведенные в статье фрагменты программ написаны на языке Pascal. Предполагается некоторое знакомство читателя с принципами работы системы Turbo Vision.

Базовые объекты

Конечно, при разработке многооконного интерфейса надо использовать объекты. Структура интерфейса может быть сходной со структурой известной объектно-ориентированной системы Turbo Vision фирмы Borland, но речь о разработке независимой системы, а не о модификации Turbo Vision.

Базовым объектом многооконной системы является, естественно, окно. Окно занимает прямоугольную область экрана и полностью отвечает за ее изображение. Окно также должно быть способно реагировать на те или иные поступающие извне события (нажатия клавиш на клавиатуре, кнопок мыши, различные команды и т.п.). Самостоятельно осуществлять выборку событий (и опрос клавиатуры в том числе) окно не должно. Для изображения окна на экране и для обработки событий используются виртуальные методы Show и HandleEvent:

```
Type  PWindow = ^TWindow.

 TWindow = Object
    A,B . TPoint; { record X,Y : Word, end; }
    Next : PWindow, { следущее окно в списке }
    Owner : PWindow, { адрес владельца }
    constructor Init(X1,Y1,X2,Y2 : Word);
    procedure Show(), virtual;
    procedure HandleEvent(var Event : TEvent); virtual;
 End;
```

Управление окнами осуществляется с помощью администратора многооконной системы — объекта типа TProgram. Этот объект содержит ряд методов, обеспечивающих выборку событий и обработку этих событий совместно с открытым в данный момент окном. TProgram также выполняет все действия, связанные с формированием экранного изображения. Поскольку экран — разделяемый ресурс, окна не должны самостоятельно обращаться к нему и все операции вывода на экран должны происходить с участием администратора системы (окно посылает сообщение — администратор выполняет необходимые действия). Объект TProgram также является окном и выглядит примерно так:

```
Type TProgram = Object (TWindow)

      constructor Init;
      procedure Open(P : PWindow); { открытие окна }
      procedure View(P : PWindow); { изображение окна }
      procedure Hide(P : PWindow); { закрытие окна }
 ...
      procedure GetEvent(var Event : TEvent);
      procedure HandleEvent(var Event : TEvent); virtual;
      procedure Exec; { цикл обработки событий }
  End,
```

Организация вывода на экран

Для формирования изображения администратор многооконной системы должен обеспечивать выполнение следующих действий:

1) открытие новых окон и изображение их на экран;
2) вывод информации в любом из открытых окон (независимо от его положения относительно других окон);
3) закрытие окон (удаление окон с экрана;

4) перемещение окон по экрану и, возможно, изменение их размеров.

Наиболее сложное из них — вывод информации в окне. В соответствии с принятыми правилами это действие сводится к обновлению изображения либо всего окна, либо его части (если эта часть сама является окном).

Итак, пусть указателю P присвоен адрес окна и требуется обновить изображение этого окна. Если окно активно (и, следовательно, расположено выше других окон), больших проблем не возникает. Иначе, если окно хотя бы частично закрыто другими окнами, изображение этих окон будет повреждено и администратору системы придется как-то восстанавливать его. Проще всего нарисовать поврежденные окна заново:

```
X1:=P^.A.X; Y1:=P^.A.Y;
X2:=P^.B.X; Y2:=P^.B.Y;
P^.Show(); P:=P^.Next;
while P<>nil do
begin
 if (X1<=P^.B.X) and (P^.A.X<=X2) then
 if (Y1<=P^.B.Y) and (P^.A.Y<=Y2) then P^.Show();
 P:=P^.Next;
end;
```

Здесь мы учли, что надо восстанавливать лишь те окна, что закрывают данное. Вопрос о параметрах метода Show мы сейчас обсудим.

Отметим важный момент. Выполнять эту процедуру непосредственно на экране нежелательно. Хотя она и приведет к правильному результату, ее выполнение будет сопровождаться неприятными эффектами на экране. Формировать изображение надо либо на невидимой странице видеопамяти, либо просто в оперативной памяти (в буфере изображения). На экран должен выводиться лишь результат этих действий. Использование невидимой видеостраницы имеет ряд преимуществ, однако этот вариант не универсален — не все видеоконтроллеры поддерживают более одной страницы изображения. Поэтому мы будем ориентироваться на второй вариант.

Буфер изображения удобно организовать в виде объекта, содержащего подобный видеопамяти массив (или ссылку на него) и набор методов для работы с этим массивом (как минимум, рисование линий, прямоугольников и вывод строк текста). Также должен быть предусмотрен метод, копирующий заданную прямоугольную область из буфера изображения в видеопамять.

В случае, если предполагается использовать графический режим 12H (VGA, 640x480x16), объект TVideoBuff должен содержать ссылки на четыре битовые плоскости:

```
Type  PVideoBuff = ^TVideoBuff;

  TVideoBuff = object
     PMem : array [0..3] of Pointer;
     procedure init;
     procedure Fill(X1,Y1,X2,Y2 : Word; Color . Byte);
     procedure Line(X1,Y1,X2,Y2 : Word; Color : Byte);
     procedure PutS(X,Y : Word; St : String; Color : Byte);
     ...
     procedure Copy(X1,Y1,X2,Y2 : Word);
   End;
```

Для того, чтобы сформировать изображение, администратору потребуется один экземпляр объекта типа TVideoBuff. Ссылка на этот экземпляр будет передаваться окнам как параметр метода Show:

```
procedure TWindow.Show(PV : PVideoBuff); virtual;
```

Отметим, что размещение экземпляра объекта TVideoBuff потребует 153 600 байт оперативной памяти. Обычно это возможно, но с отладкой программы могут возникнуть трудности. Для их преодоления следует использовать видеорежим, не требующий столько памяти (конечно, только на время отладки). Ниже мы укажем, как это сделать.

Использовать функции BGI, к сожалению, не удастся и придется всю графику написать заново. Упрощает положение то, что для создания интерфейса не нужно большого количества функций.

Работа с активным окном в принципе не требует использования буфера изображения, но и здесь буфер полезен — например, он улучшит выполнение действий типа прокрутки текста в окне. К тому же для прямого вывода на экран вам потребуется две графические библиотеки — объект TVideoBuff и библиотека обычных функций, например BGI.

Открытие нового окна

Эта процедура сводится к включению окна в список (или перемещению окна на вершину списка, если окно уже находится в нем) и изображению этого же окна на экране.

Закрытие окон

Для простоты рассмотрим только процедуру удаления с экрана самого верхнего окна. Она сводится к последовательному изображению всех окон, хотя бы частично закрытых активным окном. Поскольку все действия производятся в буфере изображения, каких-либо неприятных эффектов вы не увидите. Вот эти действия (P указывает на активное окно):

```
X1:=P^.A.X; Y1:=P^.A.Y;
X2:=P^.B.X; Y2:=P^.B.Y;
P:=@Self;
while P^.Next<>nil do
begin
 if (X1<=P^.B.X) and (P^.A.X<=X2) then
 if (Y1<=P^.B.Y) and (P^.A.Y<=Y2) then P^.Show(@VideoBuff);
 P:=P^.Next;
end;
Copy(X1,Y1,X2,Y2);
```

Перемещение окон по экрану и изменение их размеров

Из-за невысокого быстродействия ПЭВМ эти действия не могут выполняться так же, как они выполняются системой Turbo Vision в текстовом режиме. Поэтому придется ограничиться перемещением по экрану контура окна, закрытием окна на старом месте и открытием этого же окна на новом месте.

Одним из важнейших показателей качества графического интерфейса является скорость формирования изображения. Естественно, достичь быстродействия текстового режима невозможно, и использование буфера изображения еще больше снижает быстродействие системы. Однако оно остается приемлемым. В табл. 1 приведено время выполнения основных графических операций. Для сравнения включены данные о быстродействии аналогичных методов объекта TVideoBuff экранных функций (вторая колонка) и функций Borland Graphic Interface. В табл. 2 указано

время построчного копирования различных прямоугольных областей из буфера изображения в видеопамять. Все данные получены на ПЭВМ PC/AT с тактовой частотой 12 МГц (CPU speed 550% — PC Tools 5.0) и VGA-монитором (режим 12Н, 640x480x16). Время указано в миллисекундах.

Таблица 1. Время выполнения основных операций

	TVideoBuff	RTL	BGI
Bar (3, 3,636,476)	60	60	70
Bar (3, 3, 12,476)	19	8.0	17
Bar (3, 3,636, 12)	1.4	1.4	1.6
Line (3, 3,636,476)	25	14	8.0
Line (3, 3, 12,476)	18	10	5.0
Line (3, 3,636, 12)	25	14	6.0
Rect (3, 3,636,476)	12	6.0	20
Rect (3, 3, 12,476)	12	5.0	10
Rect (3, 3,636, 12)	1.2	0.8	12
OutTextXY (3,3,'TEXT')	2.6	1.0	3.6
Шрифт	(8x14)	(8x14)	(8x8)

Таблица 2. Время копирования изображения

Copy(3, 3,636,476)	300
Copy(3, 3, 12,476)	60
Copy(3, 3,636, 12)	6.0

Из табл. 1 видно, что несмотря на отсутствие аппаратной поддержки, скорость выполнения основных графических операций в буфере изображения соизмерима со скоростью обычной графики (это объясняется невысоким быстродействием видеопамяти). Низкая по сравнению с BGI скорость рисования линий связана с несовершенством алгоритма и, видимо, может быть увеличена.

Скорость копирования изображения из буфера в видеопамять не очень велика. Поэтому быстродействие системы существенно зависит от аккуратности программирования. В частности, все окна следует разбивать на окна меньшего размера, чтобы при изменении одного из вложенных окон не переписывать содержащее его окно целиком. (Кстати, при работе в текстовом режиме такие действия вполне допустимы.) Заметим также, что использование второй страницы видеопамяти (если она есть) могло бы ускорить эту операцию. В частности, контроллер EGA позволяет копировать четыре байта за один цикл.

Скорость выполнения таких операций, как закрытие окна, снижается пропорционально увеличению числа открытых окон. И здесь можно посоветовать лишь не открывать слишком много окон одновременно и на время выполнения длительных операций придавать курсору мыши форму песочных часов (как в Windows).

Опишем также более эффективный алгоритм изображения неактивных окон. Он не требует обновления окон, лежащих выше отображаемого и состоит в следующем. Используется дополнительная битовая плоскость (плоскость 4 в случае EGA/VGA), биты которой определяют допустимость копирования соответствующих точек данных из буфера изображения в видеопамять. Для того, чтобы записать данные в нее, создается метод Mask(X1,Y1,X2,Y2:Word; H:Byte). Теперь достаточно изобразить в буфере нужное окно и установить соответствующим образом биты дополнительной плоскости:

```
X1:=P^.A.X; Y1:=P^.A.Y, X2:=P^.B.X; Y2:=P^.B.Y;
P^.Show(@VideoBuff); Mask(X1,Y1,X2,Y2,$FF);
P:=P^.Next;
while P<>nil do
begin
 If (X1<=P^.B.X) and (P^.A.X<=X2) then
 if (Y1<=P^.B.Y) and (P^.A.Y<=Y2) then
 Mask(P^.A.X,P^.A.Y,P^.B.X,P^.B.Y,$00);
 P.=P^ Next;
end;
Copy(X1,Y1,X2,Y2);
```

Совместимость графических режимов

Как известно, текстовый режим (80x25 символов) поддерживают все видеоконтроллеры. При этом использование более совершенной аппаратуры автоматически улучшает качество изображения. С графикой дело обстоит сложнее (и это естественно) — имеется несколько режимов с различным разрешением и различным количеством доступных цветов. Организация видеопамяти и программирование основных операций также могут различаться. Правда, аппаратная совместимость снизу верх имеется, но это не выход из положения — странно использовать VGA-монитор в CGA-режиме.

Очевидно, объект TVideoBuff не может быть совместимым. Для перехода в другой графический режим TVideoBuff придется заменить (или, в лучшем случае, модифицировать). Используемые для вывода строк текста шрифты также должны быть различны, но лучшим решением является, по-видимому, использование векторного шрифта. В остальном же совместимость может быть достигнута.

Поскольку не предполагается какой-то определенный графический режим, при инициализации окон нельзя использовать координаты реальных точек экрана (назовем их абсолютными). Координаты окон необходимо задавать в долях размера экрана (в относительных координатах). При этом удобно считать, что монитор имеет очень высокое, но фиксированное разрешение (например, 10 000x7500 точек).

В дальнейшем относительные координаты должны быть преобразованы в абсолютные. Формулы преобразования несложны:

```
XAbs = Round (XMax*X/9999), 0<=X<=9999
YAbs = Round (YMax*Y/7499), 0<=Y<=7499
```

Константы XMax и YMax определяют соответственно горизонтальное и вертикальное разрешение монитора.

В принципе, это преобразование можно выполнить при инициализации окон (в конструкторе TWindow.Init), но в этом случае надо либо передавать каждому конструктору параметры Xmax и Ymax, либо сделать эти параметры глобальными переменными. Первое не совсем удобно, а второе делает базовый объект TWindow аппаратно-зависимым. Поэтому представляется более корректным поместить параметры Xmax и Ymax внутрь объекта TVideoBuff и производить преобразование координат при каждом отображении окна (то есть при каждом выполнении метода Show). Это не связано с

большими затратами времени и улучшает внутреннюю структуру программы — теперь вся информация о свойствах конкретного монитора размещается внутри объекта TVideoBuff.

Объект TWindow теперь должен содержать два набора координат — относительные и абсолютные:

```
Type TWindow = object
     A,B : TPoint; { относительные координаты }
     C,D : TPoint; { абсолютные координаты }
     ...
End;
```

Получив в свое распоряжение буфер изображения и зная свои относительные координаты, окно должно быть способно корректно изобразить себя в нем. Если речь идет об объектах типа окон и меню, сделать это довольно просто. Изображение различных графиков и диаграмм также несложно. И именно эта простота лежит в основе аппаратной независимости объектов оконного интерфейса.

При инициализации окон надо учитывать, что преобразование относительных координат в абсолютные производится с небольшой погрешностью (1/2 размера пиксела). Поэтому две различные, но близко расположенные в относительных координатах точки в абсолютных координатах могут совпадать. Для определения взаимного расположения окон (например, в процедуре закрытия окна) необходимо использовать их абсолютные координаты.

Теперь о цвете. Элементы многооконного интерфейса могут быть изображены в двух цветах — черном и белом (хотя, если цвет есть, отказываться от него не стоит). Поэтому можно ограничиться включением в объект TVideoBuff таблицы цветов и назначением каждому элементу изображения (рамкам, фону и заголовку окон, выделенному и невыделенному тексту и т.д.) элемента этой таблицы. Таблица цветов должна заполняться при инициализации программы. А так как для некоторых элементов интерфейса (например, для кнопок) наличие цвета желательно, имеет смысл также ввести специальный признак монохромного монитора и изображать такие объекты в соответствии с ним.

Если вы хотите обеспечить работу программы (EXE) с несколькими различными мониторами, можно использовать механизм виртуальных функций. Для этого нужно сделать TVideoBuff абстрактным объектом, а все его графические методы сделать виртуальными. Для каждого типа монитора создается производный от TVideoBuff объект. Нужный из них выбирается при инициализации программы.

А.Хохлов





Turbo Pascal 7.0. Взгляд со стороны

"Pergam turbare porro: ita haec res postulat"
(Плавт. "Привидение")
"Буду продолжать свои выдумки — этого требует дело" (пер. с лат.)

В КомпьютерПресс №№ 1,2'93 были помещены статьи, знакомящие читателя с новым продуктом фирмы Borland — седьмой версией языка Pascal. Их можно считать "взглядом изнутри".

Автор данной статьи по образованию химик, а по сфере приложения сил — преподаватель ВУЗа. Отсюда такое название статьи.

Попробуем на простом примере доказать, что ввод в Turbo Pascal 7.0 (далее TP 7.0) процедур break и continue — это только полшага в сторону повышения гибкости управляющих конструкций этого языка программирования.

На листинге 1 приведена QBasic-программа поиска корня алгебраического уравнения методом Ньютона (касательных). Почему мы начали с QBasic, хотя статья посвящена языку Pascal? Дело в том, что язык QBasic кроме традиционной тройки циклов (цикл с предпроверкой, цикл с постпроверкой, цикл с параметром) имеет и цикл с выходом из середины: DO [...] IF ... THEN [...] EXIT DO [...] LOOP. Эта конструкция наряду с другими преимуществами, о которых будет сказано ниже, позволяет реализовывать алгоритмы в их естественной последовательности. Так в программе 1 объявляются и создаются функции пользователя (анализируемое уравнение и его производная), запрашиваются значение начального приближения к корню и значение погрешности. После этого организуется цикл, но не традиционный, а с выходом из середины. В цикле, следуя естественному порядку алгоритма Ньютона, рассчитывается новое приближение к корню (X1), и, если оно отстает от предыдущего не более чем на величину заданной погрешности, задача считается решенной (EXIT DO). Если нет, то ведется подготовка к новому приближению (X1=X2), а цикл повторяется.

```
' Программа 1 (QBasic). Метод Ньютона-1
DECLARE FUNCTION Y (X)
DECLARE FUNCTION dY (X)
DIM X1, X2, Eps AS SINGLE
INPUT "X начальн."; X1
INPUT "Погрешность"; Eps
DO                                        ' Начало цикла
     X2 = X1 - Y(X1) / dY(X1)             ' Приближение к корню
   IF ABS(X2 - X1) <= Eps THEN EXIT DO    ' Выход из цикла
     X1 = X2                    ' Подготовка к новому приближению
LOOP                                      ' Конец цикла
PRINT "Y=0 при X ="; X2
END

FUNCTION Y (X)
  Y = X ^ 2 - 3                          ' Анализируемое уравнение
END FUNCTION

FUNCTION dY (X)
  dY = 2 * X                             ' Его производная
END FUNCTION
```

При реализации на языке Pascal этот несложный алгоритм обрастает "архитектурными излишествами", так как его приходится "запихивать" в прокрустово ложе либо цикла while, либо цикла do. Цикл while считается "главным". С него и начнем — см. листинг 2.

```
{Программа 2 (Pascal). Метод Ньютона-2}
function Y(X: real): real; begin
    Y:=X*X-3;                          {Анализируемое уравнение}
end, {end function}

function dY(X. real): real; begin
    dY:=2*X;                           {Его производная}
end; {end function}

var X1, X2, Eps: real;
begin
   write('X начальн. ? '); readln(X1);
   write('Погрешность ? '); readln(Eps);
   X2:=X1+2*Eps;                      {Обманываем цикл "пока"}
   while abs(X2-X1) > Eps do begin    {Начало цикла "пока"}
     X1:=X2;                   {Подготовка к новому приближению}
     X2:=X1-Y(X1)/dY(X1);             {Приближение к корню}
   end;                               {Конец цикла "пока"}
   writeln('Y=0 при X =', X2);
end.
```

Цикл с предпроверкой требует, чтобы булево выражение заголовка было определено еще до входа в цикл, а этого нет при поиске корня методом Ньютона. Приходится до входа (и для входа) в цикл писать X2:=X1+2+Eps. Подобным образом иногда лгут детям (а машина — тоже дитя), заставляя их что-то делать или что-то не делать. Строку X2:=X1+2+Eps можно уподобить стартеру двигателя внутреннего сгорания, работающего, кстати, как и программа 2, циклически. Вот какими аллегориями — дитя, двигатель — обросла наша простенькая программа из-за того, что на языке Pascal нет цикла с выходом из середины. Можно отметить и другую ненатуральность

3. Зак. № 3781

программы 2 — "постановку телеги впереди лошади": в цикле сначала приходится готовиться к новому приближению, хотя еще не ясно, понадобится оно или нет, и только потом проводить его. Этот же недостаток перекочевал и в Pascal-программу 3. В ней от обманного "финта" X2:=X1+2+Eps удалось избавиться за счет замены "главного" цикла while на "второстепенный", но более подходящий для данного случая цикл do. Однако вылезло новое "шило из мешка" — обманывать приходится уже не компьютер, а самого пользователя, запрашивая у него значение X1, а засылая его (значение) в переменную X2.

```
{Програмна 3 (Pascal). Метод Ньютона-3}
function Y(X: real): real; begin
     Y:=X*X-3;                           {Анализируеное уравнение}
end; {end function}

function dY(X: real): real; begin
     dY:=2*X;                            {Его производная}
end; {end function}

var X1, X2, Eps: real;
begin
   write('X начальн. ? '); readln(X2);
          {а должно быть readln(X1)};
   write('Погрешность ? '); readln(Eps);
   repeat                                {Начало цикла "до"}
     X1:=X2;                   {Подготовка к новому приближению}
     X2:=X1-Y(X1)/dY(X1);                {Приближение к корню}
   until abs(X2-X1) <= Eps;              {Конец цикла "до"}
   writeln('Y=0 при X =', X2);
end.
```

Процедура break, введенная в седьмую версию Turbo Pascal, призвана вернуть программам 2 и 3 их естественность, но...

QBasic-конструкция DO ... LOOP (см. листинг 1) на листинге 4 превратилась в Pascal-конструкцию (версия TP 7.0) repeat ... if ... then break; ... until 1=2, окончание которой можно выразить латинским афоризмом "ad calendas greaces" — "до греческих календ". Автор с замиранием сердца пытался убрать в программе 4 сюрреалистическое равенство "1=2", ставя точку с запятой сразу за ключевым словом until, но компилятор при прогонке программы стал "ругаться" и требовать булева выражения. Пришлось, как и в программах 2 и 3, идти ну не на обман, так на натяжку: "выполняй, пока рак на горе не свистнет".

```
{Програмна 4 (Borland Pascal 7.0). Метод Ньютона-4}
function Y(X: real): real; begin
     Y:=X*X-3;                           {Анализируеное уравнение}
end; {end function}

function dY(X: real): real; begin
     dY:=2*X;                            {Его производная}
end; {end function}

var X1, X2, Eps: real;
begin
   write('X начальн. ? '); readln(X1);
   write('Погрешность ? '); readln(Eps);
   repeat                                {Начало цикла}
        X2:=X1-Y(X1)/dY(X1);             {Приближение к корню}
     if abs(X2-X1)<=Eps then break;      {Выход из цикла}
        X1:=X2;                  {Подготовка к новому приближению}
   until 1=2;                            {Конец цикла}
   writeln('Y=0 при X =', X2);
end.
```

История с вводом в Pascal функции break подтверждает старую истину о том, что "нет ничего практичнее хорошей теории". И вот почему.

Вышеприведенный анализ циклов на языке Pascal имеет не только сугубо практический, но и чисто теоретический аспект. Как известно, набор управляющих конструкций этого языка ведет свою родословную от основной структурной теоремы Э.Дейкстры, объявившей войну меткам: "Алгоритм любой сложности можно реализовать, используя только цикл while и альтернативу". Автор затратил уйму времени и сил в поисках статьи или книги с доказательством этой теоремы, но так и не нашел его. А вот показать, что эта теорема неверна, можно в два счета — см. листинги 5 и 6. Эти программы решают уже известную нам задачу о корне алгебраического уравнения, но другим методом — методом половинного деления. Его алгоритм — это цикл приближения к корню с вложенной альтернативой (т.е. простейшая иллюстрация теоремы Дейкстры): если корень правее центра интервала А-В, то к нему (к центру) подтягивается левый край (A:=X); если нет — правый (B:=X). В программе 5 альтернатива (if ... then A:=X else B:=X;) заменена на два цикла while, операторы тела которых попеременно выполняются либо раз, либо ни разу. В аналогичной QBasic-программе (листинг 6) также обошлись без альтернативы. Более того, удалось избавиться и от булевой переменной-диспетчера Flag, заменив два цикла while на один цикл, но с двумя выходами из середины — одним условным, а вторым безусловным.

```
{Програнма 5 (Turbo Pascal 7.0). Метод половиниого деления-1}
function Y(X: real): real; begin
     Y:=X*X-3;                      {Анализируеное уравнение}
end; {end function}

var A, B, Ya, X, Eps: real; Flag: boolean;
begin
  while 2*2=4 do begin  {Имитация цикла с выходом из середины}
     write('A ? '); readln(A);
     write('B ? '); readln(B);
   if (Y(A) * Y(B) < 0) and (B > A) then break;
     writeln('На этом отрезке даный метод неприменим!');
  end; {end loop with break}
  write('Погрешность ? '); read(Eps);
  Ya := Y(A);                  {Расчет Y на левом конце отрезка}
  while B - A > Eps do begin        {Начало цикла}
    X := (A + B) / 2;               {Делим отрезок пополам}
    {Истинная альтернатива}
    {If Ya * Y(X) > 0 then A := X else B := X;}
    Flag:=TRUE;     {Имитация альтернативы (в пику Дейкстре)}
    while (Ya * Y(X) > 0) and Flag do begin {begin then}
         A:=X;                      {Правое плечо альтернативы}
    Flag:=FALSE; end; {end then}
    while Flag do begin {begin else}
         B:=X;                      {Левое плечо альтернативы}
    Flag:=FALSE; end; {end else and if}
  end; {end while}
  writeln('Y=0 при X =', X);
end.
```

```
'Програнма 6 (QBasic). Метод половииного деления-2
DECLARE FUNCTION Y (X)
DIM A, B, Ya, X AS SINGLE
DO          ' Цикл с выходом из середины со "словами прощания"
  INPUT "A, B"; A, B
  IF Y(A) * Y(B) < 0 AND B > A THEN PRINT "Все OK": EXIT DO
 PRINT "На этом отрезке метод половииного деления неприменим!"
LOOP                            ' Конец цикла
INPUT "Погрешность"; Eps
Ya = Y(A)                       ' Расчет Y на левом конце отрезка
DO WHILE B - A > Eps            ' Начало цикла
   X = (A + B) / 2              ' Делим отрезок пополам
   DO                         ' Имитация альтернативы на базе цикла
     IF Ya * Y(X) > 0 THEN A = X: EXIT DO   'Правое плечо
     B = X: EXIT DO             ' Левое плечо альтериативы
   LOOP                         ' Конец альтернативы
LOOP                            ' Конец цикла
```

```
PRINT "Y = 0 при X ="; X
END

FUNCTION Y (X)
  Y = X ^ 2 - 3               ' Анализируемое уравнение
END FUNCTION
```

Весь фокус программ *5* и 6 в том, что альтернатива — это средство ускоренного “путешествия“ по алгоритму только в одну сторону (сверху вниз и слева направо, если смотреть на листинг), а цикл — в обе. Отсюда и ненужность (в теоретическом плане, конечно, а не в практическом) альтернативы. Цикл же DO [...] ′ IF ... THEN [...] EXIT DO [...] LOOP можно считать гибридом цикла и альтернативы.

Доказательством этой теоремы Э.Дейкстры может служить факт, что до сих пор не было случая, когда задуманный алгоритм нельзя было бы реализовать, используя только цикл while и альтернативу. Если альтернативу исключить, то основная структурная теорема должна звучать так: “Алгоритм любой сложности можно реализовать, используя только циклы (цикл repeat ... if ... then ... break; [if ... then ... break;],,, ... until)“. Вот это-то теоретическое положение вводом процедуры break и подсовывает задним числом язык Pascal под свой фундамент. Обращаю внимание на несовершенную форму глагола — “подсовывает“, а не “подсунул“ — цикл с выходом из середины на языке TP 7.0 осуществим через насилие либо над циклом do (листинг 4), либо над циклом while (листинг *5*). В первом случае приходится лгать (1=2 — см. программу 4), а во втором, наоборот — писать в заголовке цикла какую-нибудь тривиальную истину — “Волга впадает в Каспийское море“ или “2x2=4“, как в программе *5*.

Теорему Э.Дейкстры следует “понизить в звании“ и называть леммой, т.е. вспомогательной теоремой, служащей для доказательства основной.

А теперь зададимся почти гамлетовским вопросом: “Быть или не быть в Pascal-программах еще каким-либо операторам между ключевым словом then и новой процедурой break?“. На языке QBasic, как видно из программы 6, покидая цикл, можно сказать “слова прощания“ — A=X или B=X, что существенно повышает гибкость этой структурной управляющей конструкции. В Pascal-программе вызов процедуры break может следовать либо сразу за ключевым словом then (или за ключевым словом else), либо находиться перед словом end внутри составного оператора. Так, например, будет выглядеть альтернатива программы 6, если ее перевести на Pascal:

```
repeat
  if (Ya * Y(X) > 0) then begin A:=X; break; end;
  begin B:=X; break; end;
until 1=1;
```

Если процедура break находится внутри составного оператора, то она превращается в тривиальный переход к метке и ее стоило бы по-честному назвать goto end. Но если уж быть до конца справедливым, то все ключевые слова языка Pascal, формирующие циклы и альтернативы, следует считать завуалированными метками и переходами к меткам, от которых программист открещивается, перепоручая возню с ними компилятору. Здесь мы вернулись к спорам, бушевавшим лет 30 назад. Процедура break, введенная в TP 7.0, дала нам повод напомнить о них. О процедуре continue автор в статье умалчивает, не найдя более-менее подходящего примера, оправдывающего ее ввод в Pascal. Правда, при поиске автора расхолаживал тот факт, что в QBasic такой процедуры (оператора) нет. А уж язык BASIC никогда не поленится перенять удачные находки своих “коллег“.

Если читатель еще не совсем запутался, то вот какую “ясность“ в проблему минимального числа управляющих конструкций алгоритмов (суть основной структурной теоремы Э.Дейкстры) внесла переведенная с английского и выпущенная в 1989 году издательством “Мир“ книга “Языки программирования Ада, Си и Паскаль. Сравнение и оценка“. В ней на стр. 72 и 73 сказано: “В соответствии с генеалогией управляющих структур... циклы с использованием оператора завершения Break и оператора продолжения Continue относятся к классу $DREC_1$. (Такую аббревиатуру европеец не мог придумать: dreg(s) по-английски “отбросы“; die Dreck по-немецки “грязь“, “дерьмо“.) Теоремы, доказанные Р.Косараю (точно, не европеец — японец), показывают, что управляющие структуры, относящиеся к классу $DREC_1$, не могут быть эмулированы с помощью управляющих структур, относящихся к классу D (название этого класса образовано от первой буквы фамилии Э.Дейкстры — автора основной структурной теоремы), которые составляются из произвольного числа условных операторов If, операторов цикла While и их конкатенаций. На самом деле управляющие структуры, относящиеся к классу $DREC_i$ (имеющие в своем составе оператор выхода Exit(i) (или break, как в TP 7.0), обеспечивающий выход на i уровней вверх, и оператор Cycle(i) (или continue), обеспечивающий выполнение цикла на i-м уровне вложенности, как, например, в языке Блисс), яв-



ляются более мощными, чем управляющие структуры, относящиеся к классу $DREC_{i-1}$. ...Однако программы, содержащие циклы с использованием операторов Exit и Cycle, оказываются намного сложнее для понимания. Анализ показывает, что необходимость использования оператора Exit возникает крайне редко. Необходимость наличия управляющих структур более высокого уровня, чем управляющие структуры класса D, до сих пор не доказана".

Вот так-то. Начали "за упокой" основной структурной теоремы, а кончили — "за здравие".

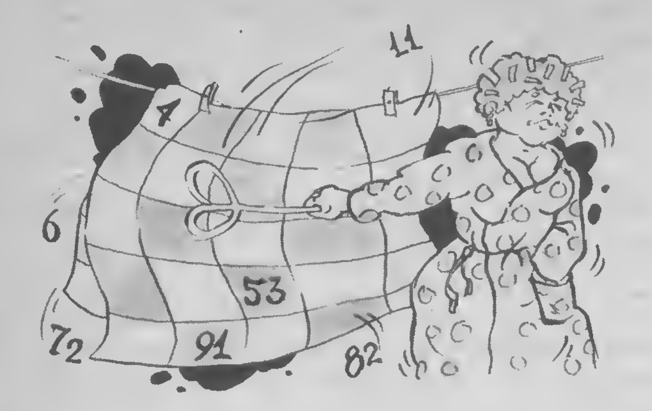
Но вернемся к практическим заботам.

Естественно, и без процедуры break можно писать циклы с любым числом выходов из середины и со "словами прощания". Мы уже разобрали реализацию цикла с выходом из середины через цикл while (листинг 2) и цикл do (листинг 3). Кроме того, часть операторов циклов while, do или for можно обойти, вставив в тело цикла альтернативу с одним плечом. Если выходов из цикла много, то его тело можно вынести в отдельную процедуру, разбросав в ней процедуры exit, которые в теле другой процедуры подобны процедуре break. Кстати, непонятно, почему break и exit в отличие, например, от if или while не удостоились звания ключевых слов. Что это еще за деление на чистых и нечистых! Это тем более непонятно, если принять во внимание, что ключевые слова do, then, else и begin вообще нельзя считать ключевыми словами, так как они не несут никакой смысловой нагрузки, выполняя чисто декоративные функции комментариев, украшающих программу и делающих ее более "читабельной". С другой стороны, ключевому слову end уже давно пора "развалиться на куски" — на ключевые слова end while, end for, end then, end else и т.д., что исключит перегруженность текстов программ комментариями типа {end while}, {end for} и т.д. Но на изменение списка ключевых слов языка Pascal наложено табу, нарушать которое позволительно только по "большим праздникам", например в момент ввода в Pascal элементов объектно-ориентированного программирования (Turbo Pascal *5.5*). Возведение же процедур exit и break в ранг ключевых слов было бы не праздником, а печальным событием, напоминающим о том, что Pascal базируется на дефектной основной структурной теореме.

Теперь взглянем на седьмую версию Pascal глазами программиста-прикладника, который, конечно, очень далек от теоретических споров, тем более тридцатилетней давности, и закончим статью о структурных управляющих конструкциях Pascal структурным анализом ее заголовка: "взгляд со стороны".

1. Почему со стороны?
   1.1. Автор по образованию химик, см. начало статьи.
   1.2. Автор для решения своих прикладных задач использует BASIC (Quick- и Q-версии от Microsoft).
   1.3. Автор пытается перейти на Pascal и C, но... "рад бы в рай, да грехи не пускают". Чьи грехи — см. п. 2.4.
2. Какой взгляд?
   2.1. Пристрастный, так как:
      2.1.1. См. п. 1.2.
      2.1.2. Статей, просто информирующих о новинках Turbo Pascal, и так достаточно.
   2.2. Поверхностный, так как:
      2.2.1. См. п. 1.1.
      2.2.2. Автор не работал ни с документацией, ни с дистрибутивом TP 7.0, а только с его пиратской копией, так как...
   2.3. Объективный, так как автор не имеет личного интереса преувеличивать или преуменьшать достоинства того или иного языка программирования.
   2.4. Конструктивный, так как статья завершается списком необходимых доработок языка Pascal для того, чтобы к нему повернулись программисты-прикладники. В числе таких доработок:
      2.4.1. Наличие цикла с выходом из середины.
      2.4.2. Наличие оператора возведения в степень.
      2.4.3. Возможность организации цикла с действительным параметром, а не только с целочисленным, имеющим шаг либо плюс 1, либо минус 1.
      2.4.4. Возможность использования в именах переменных (функций и процедур) спецсимволов — греческих и русских букв, плюсов-минусов, индексов и т.д., как, например, в языке FRED, входящем в состав интегрированного пакета Framework той же фирмы Borland.
      2.4.*5*. Перенос функций и процедур "в зазеркалье" основной программы, как это сделано в языке QBasic, где после вызова программы на дисплее появляется только текст основной программы, а все процедуры и функции "не мешаются под ногами" и при необходимости вызываются на дисплей по одной, а не все сразу.
      2.4.6. Разработка укороченной версии языка TP (условно назовем ее TPascal) и, возможно, включение ее в состав операционной системы компьютера в целях исключения работы с пиратскими копиями (см. п. 2.2.2). Пример — укороченная версия языка QuickBASIC под названием QBasic, входящая в состав MS-DOS *5*.0.
      2.4.7. Включение в состав среды программирования полноценного текстового процессора, позволяющего в текстах программ менять шрифт, цвет, отмечать индексы и степени.
      2.4.8. Исключение из текстов Pascal-программ точек с запятой и ритуальной финишной точки. Неужели компилятор без помощи программиста не может разобраться, что на этих местах ничего другого стоять не может?!
3. Если кого-то "взгляд со стороны" автора не устраивает, он может смотреть со своей стороны, так как автор не настаивает, что стоит на *той* стороне.

В.Очков



CA-SuperCalc.

Удивительное Прошлое, Большое Настоящее и Великое Будущее

Одной из первых программ, разработанных для имитации работы с таблицами на компьютере, была программа SuperCalc фирмы Sorcim. Появилась она в начале 1981 года, автором исходного кода был Гэри Бэлайзн. Персонального компьютера фирмы IBM тогда в природе не существовало, и новая электронная таблица предназначалась для 8-битных персоналок[1].

С тех пор в табличном мире многое изменилось. Так, фирма Software Arts, ставшая в 1979 году пионером в области "таблицестроения" с программой VisiCalc, была куплена корпорацией Lotus — новым законодателем табличных мод. На пакете SuperCalc тоже сменился копирайт, и с тех пор он гласит: Computer Associates. Рынок spreadsheets запестрел новыми торговыми марками — Excel, Quattro...

Надо сказать, что в умах и сердцах людей, так или иначе связанных с электронными таблицами, — от производителей до конечных пользователей — SuperCalc занимает достаточно двойственное положение. Версию 5.0 этой программы одновременно назвали в двух разных журналах "устаревшим программным продуктом" и "системой будущего". Судя по результатам различных зарубежных опросов, с SuperCalc работает не более 1% пользователей. Между тем, почти любой из нас знаком с той или иной версией SuperCalc, хотя бы в лице одного из "отечественных" табличных процессоров: Абак, Варитаб, Дракон, являющихся на самом деле далеко не идеальными адаптациями SuperCalc.

Так или иначе, фирма Computer Associates честно делает все от нее зависящее, чтобы угодить имеющимся потребителям и привлечь новых. Вызвавший столько споров SuperCalc 5.0 был радикально переделан, и уже с версией 5.1 пользователи получили многое из того, о чем они до сих пор только мечтали. Однако истинный качественный скачок происходит на наших глазах, а первым его воплощением является не что иное, как...

SuperCalc 5.5

Как и прежде, система управления электронными таблицами с этой торговой маркой предъявляет минимальные требования к техническому оснащению вашего персонального компьютера. При самой что ни на есть аскетичной конфигурации — 8088 процессор, 512 Кбайт оперативной памяти, 3,2 Мбайт на жестком диске для размещения служебных файлов, любой монитор с 80-значной строкой и DOS не ниже 3.0 — вы можете использовать SuperCalc почти во всей его красе. Для нормальной работы неплохо бы иметь все 640 Кбайт ОЗУ, а еще программа может хранить данные в отображаемой памяти (спецификации LIM 4.0 — до 32 Мбайт). SuperCalc может работать в локальных сетях IBM PC, IBM Token Ring, 3COM, Novell Advanced Netware, Banyan, AT&T STARLAN, Novell Netware Lite, Lantastik, Chosen/Promise LAN.

Система SuperCalc поддерживает практически все возможные типы графических видеоадаптеров, принтеров и плоттеров. Данная версия программы распространяется у нас в локализованном — русифицированном варианте. Вся мощь SuperCalc — на изготовление полноценных русскоязычных таблиц и графиков! И никаких опасений относительно качества адаптации — программа прошла через руки российских специалис-

	V	W	X	Y	Z	AA	AB	AC
[illegible]	.044	4.009	6	84.99	35.00	55.18	9.74	
[illegible]	.033	4.01	6	84.53	36.02	54.00	9.9	
7	.056	4.002	6	90	41.01	53.09	5.9	
[illegible]	.043	4.01	6	87.43	38.04	53.47	7.69	
9	.038	4.009	6	86.39	35.78	55.48	8.74	
10	.046	4.005	6	90.59	39.56	54.75	5.69	
11	.052	4.01	6	88.83	40.97	51.96	7.06	
12	.057	4.003	6	90.4	40.69	53.62	5.69	
13	.067	4.004	6	88.6	36.79	56	7.21	
14	.069	4	6	89.56	41.6	52.3	6.1	
15	.066	4.003	6	88.42	40.87	52.28	6.84	
16	.055	4.005	6	86.85	39.59	52.47	7.94	
17	.073	3.999	6	90.31	41.6	52.75	5.66	
18	.058	4.009	6	87.06	41.27	51.6	7.13	
19	.034	4.01	6	86.4	39.73	52.08	8.2	
[illegible]	.045	4.004	6	87.2	38.58	53.56	7.87	
[illegible]	.04	4.011	6	85.32	39.06	51.99	8.95	
[illegible]	.053	4.01	6	85.63	40.47	50.97	8.56	
[illegible]	.031	4.016	6	87.25	40.56	51.06	7.58	
[illegible]	.041	4.009	6	88.49	39.56	53.48	6.96	

```
Arrange  Blank  Copy  Delete  Edit  Format  Global  Insert  Justify  Load  Move
Name  Output  Protect  Quit  Save  Title  Unprotect  Window  Zap  1-2-3  /more
  2>/
MENU  Use the alternate Lotus 1-2-3 menu for the next command     CAPS NUM SCROLL
```

[1] В дальнейшем эта система успешно применялась на Robotron-1715.

тов высочайшей квалификации, и к тому же всесторонне оттестирована на "исторической родине", то бишь в Штатах. В комплекте поставки — Руководство Пользователя, также на хорошем русском языке.

Впрочем, довольно слов — пора за дело! Берем дискету номер 1, запираем дисковод и командуем 'install'. При полном варианте установки SuperCalc займет на вашем винчестере около 5,3 Мбайт — с обучающей программой, файлами-образцами и добавочными графическими драйверами для вывода изображений на слайды. Однако и после этого две дистрибутивных дискеты (из тринадцати DS/DD) останутся невостребованными программой-установщиком — о них чуть позже.

Если при вашем компьютере живет мышь, в память загружен соответствующий драйвер, а вы сами — бывалый пользователь SuperCalc, то при входе в программу вас приятно удивит появление на дисплее "мышиного" прямоугольника. Если же вы новичок, то вас скорее удивит известие о том, что в ранних версиях программы такого замечательного сервиса не было, и не было с правой стороны экрана полоски Toolbar, в которую можно "ткнуть" мышью, не копаясь в недрах меню. И вы с удовольствием будете осваивать эту программу, находя в ней новые и новые достоинства — средства настройки цветов меню, рабочего поля и прочего, режим автосохранения данных, и конечно — возможность предварительного просмотра отчетов перед выводом на печать. Preview теперь покажет вам в точности то, что пойдет на принтер — со всеми шрифтами, оттенками и так далее.

Впрочем, обсуждая те черты SuperCalc, которые стали доступны лишь в самых последних реализациях, мы, пожалуй, забегаем вперед. Из "классических" же достоинств этой электронной таблицы прежде всего следует упомянуть простоту освоения. Документация к программе составлена весьма мудро — прочитавший за компьютером начальные главы получает основные навыки для дальнейшей деятельности. Очень быстро можно научиться редактировать табличные данные и производить вычисления. Не составит труда и создание диаграмм, тем более что теперь в SuperCalc доступен режим "быстрого рисования" для определенного блока данных. Благодаря этому потрясающие графические возможности программы (более сотни разновидностей двух- и трехмерных рисунков!) становятся еще более привлекательными. Достаточно легко в системе SuperCalc производятся операции поиска и выборки необходимой информации, а также создания печатного отчета. Хорошим подспорьем как для начинающего, так и для опытного пользователя является стандартная для всех развитых программных продуктов возможность отмены последней операции редактирования — Undo.

Следующий шаг к профессиональному овладению программой SuperCalc — создание собственных макросов для автоматизации рутинной работы, такой как ввод данных. Но истинную силу SuperCalc вы узнаете, попробовав управление "трехмерными" данными (имеется в виду возможность объединения до 255 таблиц или табличных "страниц" при одновременной работе с одной, двумя или тремя таблицами на экране). При этом ни одна электронная таблица не может похвастаться такой великолепной организацией работы в многотабличном режиме! Можно связывать таблицы (как SuperCalc, так и Lotus 1-2-3), находящиеся в одном или многих файлах.

Факт наличия в SuperCalc таких мощных и гибких средств отмечали и те, кто считал этот программный продукт "устаревшим" по сравнению, например, с таблицами 1-2-3. Кстати, если вы умудрены опытом работы с 1-2-3, то переход в новую систему будет максимально безболезненным — SuperCalc поддерживает формат файлов, макросы и само меню 1-2-3. Старайтесь, однако, привыкать к "родному" меню, поскольку совокупность опций меню 1-2-3 далеко не исчерпывает возможностей системы SuperCalc.

Никак нельзя пренебречь издательским сервисом программы. Заметим, что список параметров настройки печатного вывода пополнился, например, возможностью задания размера печатной страницы, а для лазерных принтеров с двусторонней печатью реализован режим дуплекса. С пакетом по-прежнему поставляется утилита SIDEWAYS, с помощью которой чересчур широкие отчеты можно при выводе на печать повернуть на 90 градусов.

Однако не пора ли нам разобраться с двумя "лишними" дискетами дистрибутива? На одной из них тоже есть программа Install! Так-так, посмотрим...

Silverado

Давайте спроектируем идеальную СУБД. Что требуется от базы данных? Услуги ввода и редактирования данных, поиска и сортировки информации, создания отчетов — а как же иначе? Необходим набор вычислительных функций — чем больше, тем лучше. Если СУБД называется реляционной, то она, как минимум, должна обеспечивать связывание таблиц по ключевому полю. Что бы еще пожелать? Ага, вот чего нет ни в dBASE, ни в Paradox — хотим копировать и переносить поля базы данных целиком или частями, выделяя требуемое "блоками" — как в текстовом процессоре! И еще —

```
      B        C                    D                     E
 1
 2
 3   First    Last                 Company               Street
 4
 5  David    Newcomer             Visu-ALL              892 Folsom St.
 6  Jenny    Zimmerman            Computer Store        255 Main St.
 7  Jeff     Kim                  Sell-a-Brake          283 N. Center St.
 8  Sandra   Winters              Sports Galore         23842 Arguello St.
 9  Janet    Huer                 Artistic FreeWay      3825 N. Third St.
10  Patty    Nakano               PR & G                235 Brannon St.
11  Phil     Gibson               Tell-In Computers     1835 Folsom St.
12  Richard  Maloney              Bank of America       2884 Market Street
13  Claude   Pierre               The Gap               8347 Haight St.
14  Bill     Sands                Wells Fargo           9328 Jones St.
15  Ed       Minton               Lloyd and Furner      238 Turk St.
16  Wayne    Chin                 A&B                   4th St.
17
18
19
20
→ TEMP1!E16                  Text="4th St.
Width: 20  Memory: 235  Last Col/Row:X17
  1>
SILVERF1:Help  F3:Names  Ctrl-Backspace:Undo  Ctrl-Break:Cancel
```

чтобы при внесении изменений в структуру базы не надо было редактировать всякие специальные таблицы! Поместил курсор на "шапку", нажал 'INS' — вставил поле, нажал 'DEL' — удалил поле...

Глупости, скажете? СУБД — вещь серьезная, не то что электронная таблица? Да, и поныне бытует мнение — использование электронных таблиц не требует специальных знаний, а вот чтобы работать с базой данных, надо быть программистом. Действительно, электронные таблицы — в отличие от СУБД — изначально создавались для менеджеров. Но "физическое" отличие spreadsheet заключается только в использовании данных, находящихся в оперативной памяти. "Отважусь" на следующее утверждение: реализовав обмен данными между электронной таблицей и диском, и следовательно, обеспечив работу с массивами данных неограниченного объема, можно получить не что иное, как идеальную пользовательскую базу данных.

Silverado 1.1 — СУБД реляционного типа, поставляемая на тех самых двух дополнительных дискетах вместе с SuperCalc 5.5 и работающая только в системе SuperCalc (при подключении через Add-in). Главные возможности этой базы данных, собственно, перечислены в первом абзаце текущего раздела. Благодаря этим возможностям (и хорошей документации) освоить Silverado "не просто, а очень просто". Кстати, неплоха и программа, обучающая работе с Silverado — в отличие от аналогичной для SuperCalc она не просто демонстрирует действие команд меню, а требует выполнения упражнений. Вместе с обучающим курсом и образцами файлов данных Silverado занимает на жестком диске 0,8 Мбайт, а при минимальной инсталляции — всего 0,4 Мбайт.

Важной особенностью новой СУБД является ее способность к взаимодействию со средой электронных таблиц. Окно просмотра файла данных Silverado с имеющейся в нем информацией при переключении в режим SuperCalc становится электронной таблицей — со всеми вытекающими из этого последствиями. Конечно, имеется возможность экспорта-импорта текстовых и DBF-файлов, а вот стоит ли немедленно ею воспользоваться, чтобы "перекачать" всю вашу информацию в новый формат и полностью перейти на Silverado — уже другой вопрос...

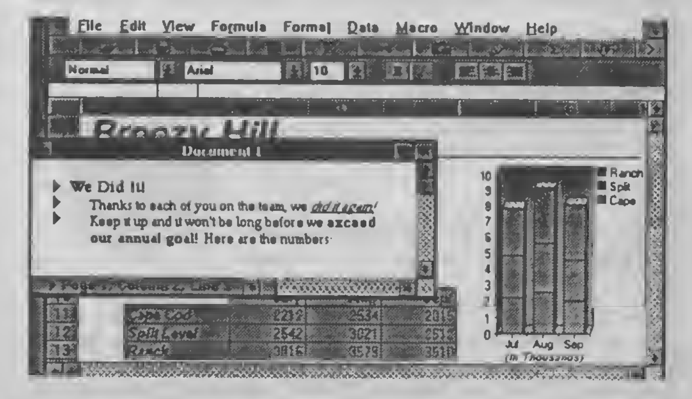
Главная проблема, которую придется решить, — проблема памяти. Руководство пользователя Silverado утверждает, что для работы СУБД требуется не менее *575* Кбайт базовой оперативной памяти. Теоретически это достижимо даже для компьютера с процессором 8086/88 (и MS-DOS *5.0*), но сможет ли кто-нибудь нормально работать, не имея ни единой TSR-программы или драйвера? Для тех, у кого персоналка классом повыше (и нет MS-DOS *5.0* или Windows 3.x), в пакет SuperCalc включена версия 2.60 драйвера HIMEM.SYS, но на практике освободить *575* Кбайт нелегко и на 286 машине с DOS, загруженной в HMA. Допустим, это удалось, и вы пытаетесь активизировать Silverado, предварительно освободив память отказом от режима Undo и возможности работы с "трехмерными" таблицами (под SuperCalc должно оставаться свободными не менее 50 000 байт). Если с первого раза все пошло как по маслу — поздравляю. В противном случае вам предстоит поистине трагическое расставание со многими любимыми вами резидентными программами. Как прошедший через все это, поделюсь небольшим опытом — на машинах с 286 процессором Silverado не загружается ни под каким видом, если в памяти находится драйвер мыши. Откровенный эгоизм — сама-то Silverado мышь не использует... Ищите виноватых среди TSR-программ и тогда, когда система "виснет". Снять все эти проблемы может лишь загрузка резидентов в Upper Memory, а для этого нужен 386...

Таким образом Computer Associates как бы сигнализирует нам: "Конечно, вы можете еще долго работать на XT, но все-таки для хороших программ нужна хорошая аппаратура". Что-что? Говорите, на 386 вы будете пользоваться Excel? Ах, вы еще не выбрали между 1-2-3 for Windows и Quattro Pro for Windows? Подумайте — может быть, все-таки...

...SuperCalc for Windows

С реализацией "оконной" версии своего любимца фирма Computer Associates явно не торопилась. Ветеран среди электронных таблиц (SuperCalc на 4 года старше пакета 1-2-3, на 6 лет — Excel, на 8 — Quattro) лишь в текущем году выйдет на тропу Windows. "Дебют", однако, впечатляет: посмотрите сами, как реализован принцип WYSIWYG при формировании составного документа, и тогда вы поймете, *что* стоит за рекламными девизами Tomorrow's Spreadsheet For Today's Computers и Software superior by design. Радующая глаз картинка — лишь внешняя сторона дела, новое качество достигается благодаря использованию метода связи и внедрения объектов (OLE).

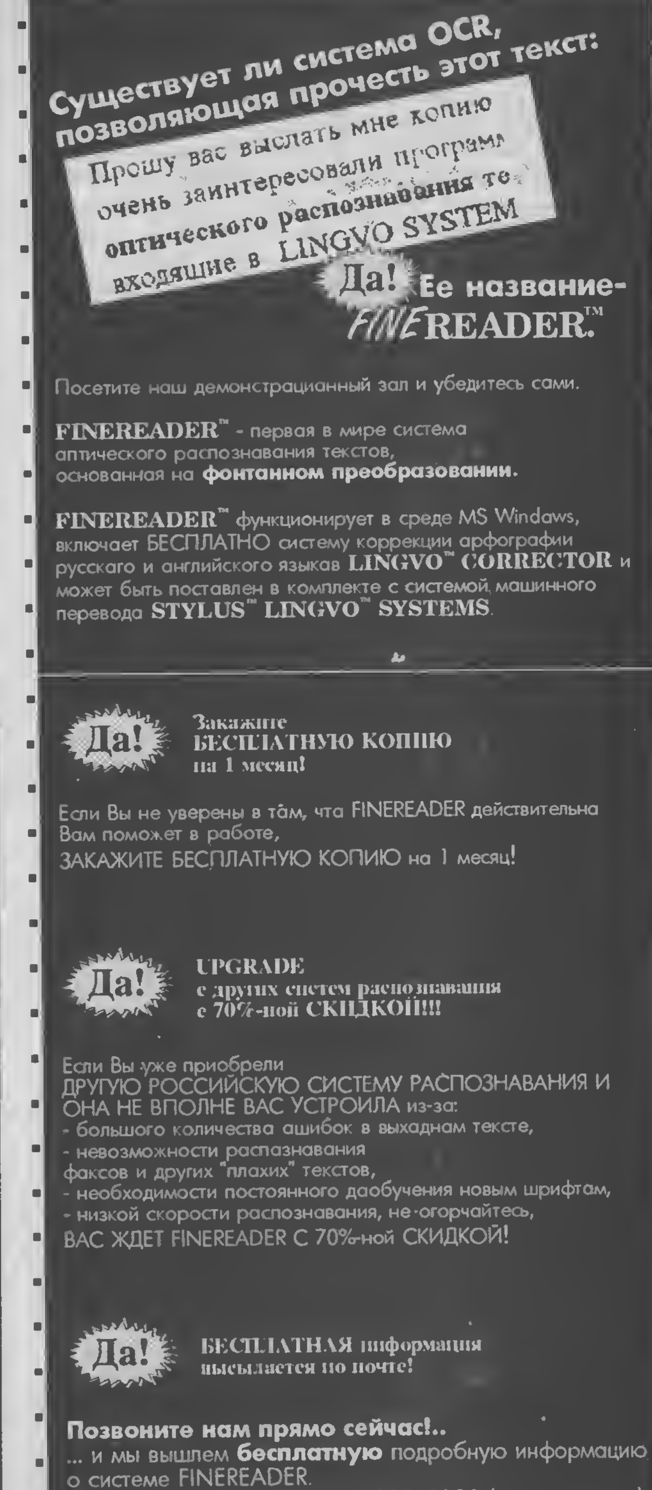
Получая новую версию любой программы, вы всегда беспокоитесь о том, сумеете ли вы без особых затруднений воспользоваться своими старыми данными. SuperCalc for Windows позволит вам работать с файлами форматов SuperCalc for DOS, 1-2-3, Excel. В отношении макроязыков SuperCalc for Windows поначалу кажется не таким уж и полиглотом, поскольку поддерживает "лишь" макросы SuperCalc for DOS и 1-2-3. Но!

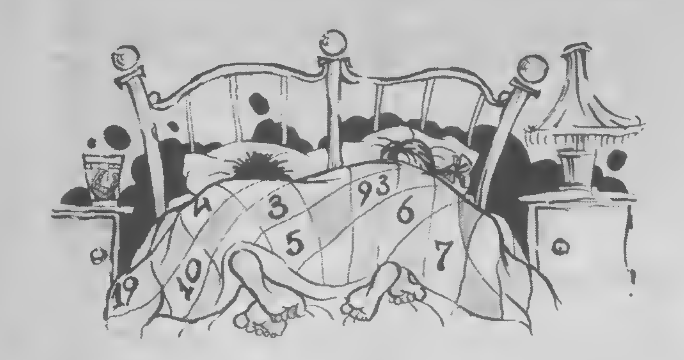


На самом деле все значительно круче, ибо отныне в вашем распоряжении встроенный в SuperCalc for Windows язык программирования CA-BL — Computer Associates Basic Language, являющийся дальнейшим развитием макроязыков электронных таблиц. CA-BL поддерживает динамически компонуемые библиотеки (DLL) и ныне является промышленным стандартом фирмы Computer Associates, обеспечивая интеграцию программ for Windows: CA-Textor, CA-dBFast, CA-Compete!, CA-SuperCalc, CA-SuperProject, а также средств бухгалтерского учета семейства ACCPAC. Связь CA-BL и вообще SuperCalc с другими Windows-приложениями осуществляется методом динамического обмена данными (DDE). Как вы уже догадались, создатели SuperCalc for Windows и здесь не забыли о совместимости — можно будет переводить на CA-BL макросы, написанные для SuperCalc for DOS, 1-2-3, Excel.

Что же на самом деле представляет собой CA-SuperCalc? "Устаревший программный продукт" или электронную таблицу нового поколения? Недорогую программу для пользователей маломощных персоналок или Tomorrow's Spreadsheet For Today's Computers? Теперь вы все знаете, так что выбор за вами!

К.Ахметов





Пожалуй, сегодня нет необходимости убеждать кого-либо в достоинствах электронной таблицы Quattro Pro, созданной фирмой Borland, как средства ведения делопроизводства, бухгалтерских расчетов, автоматизации принятия решений и постановки научных экспериментов. Наиболее эффективному использованию возможностей этой электронной таблицы посвящена книга "Quattro Pro: работаем профессионально", выдержки из которой предлагаются вашему вниманию.

QUATTRO PRO: работаем профессионально

Создание команд-акселераторов

Для наиболее часто используемых команд меню можно создать акселераторы. Выберите (курсором) команду в меню, нажмите одновременно клавиши [Ctrl] и [Enter], а затем, не отпуская [Ctrl], нажмите любую клавишу с буквой.

Quattro Pro закрепит комбинацию [Ctrl] и выбранной буквы за данной командой меню. В дальнейшем данную команду можно выполнить по нажатию этих клавиш без обращения к меню. Список созданных акселераторов держите под рукой. Удаление акселератора производится после выбора соответствующей команды меню и нажатия [Ctrl-Enter], а затем [Del] два раза. В Quattro Pro уже имеется набор акселераторов для некоторых команд меню (акселераторы расположены справа от названия команд меню).

Меню с фиксацией

Для того чтобы Quattro Pro "запоминала" маршрут до последней выполненной команды в каждом меню и помещала на нее курсор при возвращении в это меню, запишите следующий макрос в две ячейки:

```
{/ Startup;Remember}{HOME}~
{/ Defaults;Update}
```

и выполните его с помощью команды /Tools|Macro|-Execute.

Выполняйте этот макрос с каждым файлом .MU, описывающим меню, в котором вы бы хотели использовать эту особенность. Для выхода из меню с фиксацией используйте [Esc] или клавиши [Ctrl-Break] вместо Quit, иначе команда Quit будет выделена курсором при следующем обращении к этому меню.

Сцепление меток в формулах

В формулах можно делать ссылки на ячейки с текстом (метками) точно так же, как на ячейки с числовыми значениями. Для объединения текстовых значений используется знак &. Например, +"Итого "&C6 присоединяет к метке "Итого " текст из ячейки C6. (Если C6 содержит числовое значение, Quattro Pro сообщит об ошибке.) Текст в формуле, включая пробелы, должен заключаться в кавычки. Если ячейки A1, A2 и A3 содержат буквы B, C и E соответственно, результатом формулы +A1&A2&A3 будет слово "BCE".

Запомнить текущее положение

Чтобы быстро просмотреть другую часть таблицы, не покидая текущей ячейки, используйте клавишу [F5].

Обычно при нажатии [F5] Quattro Pro запрашивает ячейку, куда нужно переместиться. Вместо указания адреса ячейки с помощью клавиши-стрелки или мыши переместитесь в нужную часть таблицы, но не нажимайте [Enter] и не выполняйте никаких команд. Для возвращения в исходную ячейку нажмите [Esc].

Блокирование вставки строк или столбцов

Чтобы запретить вставку новых строк или столбцов в таблицу, не устанавливая защиту всей таблицы, запишите произвольную информацию (например, пробел) в ячейки последней строки 8192 или последнего столбца IV. Содержимое этих ячеек будет препятствовать вставке новых строк или столбцов в таблицу.

Заголовок с отступом

Чтобы столбец данных под заголовком имел отступ, вставьте пустой столбец слева от данных и уменьшите его ширину с помощью команды /Style|Column Width. Переместите заголовок на одну ячейку влево, то есть в новый столбец. Текст заголовка при этом "захватит" и исходный столбец. Данные, расположенные ниже заголовка, будут сдвинуты вправо (появится отступ) по отношению к заголовку на ширину нового столбца. Рисунок наглядно показывает таблицу, в которой использован этот прием для выделения подзаголовков в столбце.

Связь всех открытых таблиц

Используйте перед ссылками на ячейки команду связи [*], чтобы организовать ссылки на данные, расположенные в одних и тех же ячейках, но в разных таблицах. Например, с помощью формулы @SUM([*]A1) вычисляется сумма значений ячеек A1 всех открытых таблиц (исключая текущую таблицу). Это особенно удобно при создании таблицы, являющейся итоговой для нескольких однотипных таблиц (например, ежегодная таблица продаж, составляемая по таблицам ежемесячных продаж). Эти однотипные таблицы должны иметь одну и ту же форму, а в одинаковых ячейках должны быть записаны одинаковые по смыслу значения. Таблицы, подготовленные для составления итоговой таблицы, должны быть единственными открытыми таблицами (отдельно от итоговой). Чтобы ограничить связь только таблицами Quattro Pro, используйте в формулах команду связи [*.WQ1].

Имя блока для печати

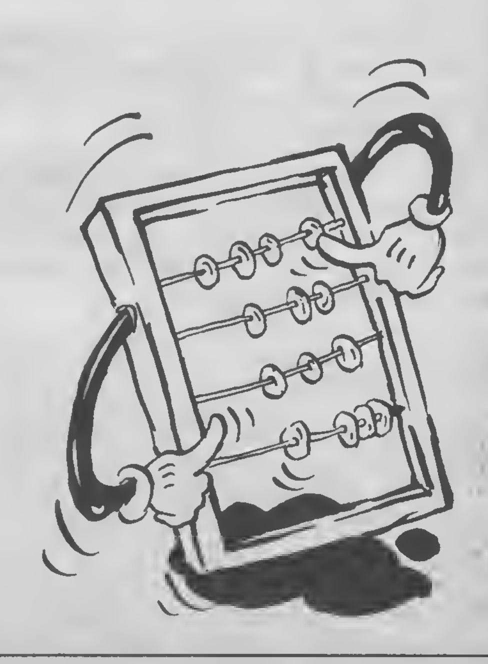
Удобным является прием, когда блоку ячеек с данными, который вы обычно печатаете, присваивается имя. Тогда при выполнении команды /Print|Block в ответ на запрос блока достаточно указать его имя. В этом случае Quattro Pro будет автоматически менять границы блока при вставке или удалении строк и/или столбцов, и печать блока будет правильной. С помощью этого же приема легче выбрать блок для печати из нескольких именованных блоков.

Исключение столбцов при печати

Если в блоке печати имеются столбцы, которые вы не хотите печатать, с помощью команды /Style|Hide Column|Hide предварительно подавите их индикацию. После печати восстановите индикацию столбцов на экране. Эта операция делает возможным при печати расположить рядом отдельные части таблицы. Если в вашей таблице имеются две области данных, отделенные друг от друга пустыми столбцами или столбцами данных, которые вы не хотите печатать, подавите индикацию этих разделяющих столбцов, а потом выполните команду печати.

Пароли для файлов

Если вы хотите ограничить доступ других пользователей к вашим табличным данным, защитите таблицу паролем. Для этого при записи файла на диск добавьте справа к имени файла пробел и букву P (латинскую). Quattro Pro запросит пароль (до 15 символов), затем предложит проверить пароль повторным набором его на клавиатуре. С этого момента пользователи (включая вас) должны будут правильно указывать пароль при открытии этого файла. Пароль чувствителен к регистру



клавиатуры, то есть пароль "BeeBop" не совпадает ни с паролем "Beebop", ни с паролем "beebop". Этот принцип делает пароль надежнее. Чтобы снять пароль с файла, откройте этот файл, а затем выполните команду /File|Save As и удалите букву P из строки запроса, после этого нажмите [Enter]. Изменить пароль можно следующим образом: выполните команду /File|Save As, удалите букву P, затем снова наберите на клавиатуре P и нажмите [Enter]. После этого в ответ на запрос введите новый пароль.

Выделение группы элементов

При работе с Аннотатором для объединения нескольких элементов в группу сначала выделите один элемент (с помощью [Tab] или [Shift-Tab]) и нажмите [Shift-F7] (или [Shift] и щелкните левой кнопкой мыши), затем выделите следующий и повторно нажмите [Shift-F7] и т.д. После выделения всех элементов нажмите [F7]. Сгруппированные элементы можно одновременно удалить или отредактировать, а также одновременно изменить их размеры.

Создание серии слайдов

Команда /Graph|Name|Slide дает возможность организовать автоматическую демонстрацию серии графиков (в виде серии слайдов). Предварительно постройте все графики, которые будут включены в эту серию. Присвойте каждому графику имя, используя команду /Graph|Name|Create. Запишите имена графиков в столбец ячеек следующим образом: по одному имени в одну ячейку и в том порядке (сверху-вниз), в котором графики должны демонстрироваться. В соседнем столбце справа рядом с именем каждого графика укажите время (в секундах) демонстрации этого графика на экране. Если необходимо организовать задержку демонстрации графика до нажатия любой клавиши (или кнопки мыши), укажите 0 (ноль) или оставьте пустую ячейку справа от имени графика.

Сортировка данных

При указании блока базы данных для сортировки не следует включать в этот блок заголовки, но обязательно включите все столбцы базы данных. Имена блоков и формулы при сортировке не обновляются, то есть после сортировки имена блоков по-прежнему соответствуют тем же самым ячейкам, что и до сортировки, а формулы вне базы данных после сортировки продолжают ссылаться на те же самые ячейки внутри базы данных. По этой причине присвойте имена только ячейкам первой строки базы данных и не пользуйтесь ссылками на базу данных из ячеек, не принадлежащих этой базе. Если вам нужна информация из базы данных, скопируйте ее в другую область таблицы.

Поиск пустых ячеек в базе данных

Для того чтобы определить местоположение пустых ячеек в базе данных, запишите в таблицу критериев следующую формулу:

```
@CELL("TYPE",LOCATION)="B"
```

где LOCATION — имя или адрес первой ячейки, начиная с которой будет выполняться поиск по критерию.

Доступ к внешним таблицам

При использовании функций, таких как @DSUM или @DMIN, для внешних таблиц (файлов Reflex, dBASE или Paradox) в качестве блока базы данных можно указать всю допустимую область (например, +[DATA.DB]A1..IV8192). После этого вам не придется каждый раз переопределять блок базы данных при добавлении или удалении записей, а лишняя область не повлияет на вычисления.

Размещение макросов

Хранить макросы предпочтительнее всего в блоке ячеек, расположенном ниже и правее данных таблицы. В этом случае удаление и вставка строк и столбцов в области данных не приведут к нарушению текста макроса. Для макросов, используемых с несколькими таблицами, лучше создать библиотеку макросов — отдельный табличный файл, предназначенный специально для хранения макросов (с помощью команды /Tools|Macro|Library|Yes). Иногда возникает необходимость работы с несколькими различными библиотеками (отдельная библиотека для определенного типа таблиц). В этом случае загружайте только ту библиотеку макросов, которая будет использоваться, другие библиотеки в это время должны быть закрыты. Для удобства в макросы можно добавить комментарии-метки, которые располагают в соседнем с макросом столбце (обычно, справа). Если вы мало знакомы с языком макропрограммирования, трудно понять назначение макроса, просмотрев его текст. Комментарии помогут вам быстрее определить, для чего данный макрос создан.

Имена макросов

Имена макросов и блоков хранятся в одном списке имен (Quattro Pro не делает различий между именами блоков и макросов). Чтобы отличать имена блоков от имен макросов, начинайте имена макросов со знака подчеркивания (например, _LOAD). В этом случае имена макросов будут расположены в начале списка имен.

Вставка макрокоманд

Чтобы использовать макрокоманды (например, {ESC}) в макросе, сначала запишите текст макроса как сможете. Затем отредактируйте макрос и вставьте в него необходимые команды и аргументы. Чтобы вставить макрокоманду автоматически, нажмите [Shift-F3], на экране появится список макрокоманд, одну из которых можно выбрать. Quattro Pro записывает ее в строку ввода, начиная с позиции курсора. После этого добавьте необходимые значения или аргументы для макрокоманды.

Быстрое именование макросов

Записывать макросы в таблицу лучше так, чтобы слева от них был пустой столбец. В этот столбец в дальнейшем можно записать имена и с помощью команды /Edit|Names|Labels|Right быстро присвоить эти имена макросам. Этот прием позволяет также избежать изменения имен блоков при вставке или удалении строк в макросе.

Вставка метки в текст макроса

Имеется возможность динамически вставлять метки в макрос. Например, данный макрос сохраняет таблицу в файле с именем, которое записано в ячейке A1:

```
{/ File;Save}{CLEAR}{A1}~
```

В этом случае ниже ячейки A1 должна быть пустая ячейка. Иначе содержимое этой ячейки будет интерпретироваться как макрос до первой пустой ячейки ниже A1.

Перемещение вниз при нажатии [Enter]

Этот простой макрос используется для перемещения курсора вниз при нажатии [Enter]:

```
{?}~{DOWN}{BRANCH THIS_CELL},
```

где THIS_CELL — имя ячейки, содержащей данный макрос.

Аналогично используются макрокоманды {LEFT}, {RIGHT} или {UP} (вместо {DOWN}) для выбора соответствующего направления перемещения.

Очистка строки ввода

В макросе, запрашивающем адрес, строку символов или имя файла, используйте макрокоманду {CLEAR} после запроса. Она очищает строку ввода от старого содержимого (если оно есть) и исключает возможность отмены команды при многократном использовании {ESC}.

Восстановление исходных значений параметров

Функция @CURVALUE возвращает текущее значение опции (параметра) команды меню только для тех команд, у которых в меню справа показаны значения опций. При вызове этой функции в качестве аргументов необходимо указать две части макроса-эквивалента команды меню, заключенные в кавычки. Например, {/Graph;NamesUse} — макроэквивалент для команды /Graph|Name|Display, тогда @CURVALUE("Graph", "Nameuse") возвращает имя текущего графика. Можно использовать @CURVALUE вместе с функцией @IF для полезной комбинации: "если текущее значение опции равно x, то выполнить y".

Используйте @CURVALUE вместе с {LET} для сохранения и восстановления исходных значений. Например, приведенный ниже макрос записывает имя исходного каталога, используемого по умолчанию, в ячейку H3, меняет каталог, записывает в него текущий файл и восстанавливает первоначальный каталог:

```
{LET H3,@CURVALUE("Defaults","Directory")}
{/ Defaults;Directory}{CLEAR}C:\QPDATA~
{/ File;Save}b~
{/ Defaults;Directory}{CLEAR}
+H3
{CR}
```

Экономия памяти

Существует несколько способов экономии памяти при работе с Quattro Pro:

- выгрузить из оперативной памяти резидентные программы;
- организовать размещение данных в таблице вдоль левого края (для строк в Quattro Pro используется меньше памяти, чем для столбцов);
- использовать префикс повторения метки (например, *) вместо повторения самих символов;
- применять функцию @SUM вместо перечисления длинного списка слагаемых в формулах;
- где возможно, вместо формул хранить их значения;
- разделить большую таблицу на несколько таблиц меньшего размера, затем связать их между собой; не открывать вспомогательные таблицы;
- запретить откатку.

Л.Дудина,
М.Михайлов

Заявки на книгу "Quattro Pro: работаем профессионально" присылайте по адресу: 454091, г.Челябинск, а/я 15887.



RISC PC — что нас ждет

Программное обеспечение

Единая операционная среда новейших ARC-компьютеров должна удовлетворять различным, зачастую противоречивым требованиям. Именно поэтому ОС UNIX, которая легко адаптируется к работе в одно- и многопользовательских режимах, поддерживает многопроцессорную работу, предоставляет широкие средства для коммуникаций, предлагается всеми производителями ARC-компьютеров (см. табл. 5).

Большой интерес пользователей вызвала объявленная фирмой Microsoft новая ОС Windows New Technology (NT).

Windows New Technology (NT)

Расширяя сферу применения своих программных средств, фирма Microsoft разработала новую версию Windows NT для использования на компьютерах, отвечающих требованиям спецификации ARC и построенных на базе микропроцессоров Intel 80386/80486 и MIPS Rx000. Таким образом, данная операционная система сможет эксплуатироваться как на "карманных" и персональных ЭВМ, так и на сетевых и вычислительных серверах. В системах распределенной обработки NT позволяет вести работу в режимах клиента и сервера, поддерживает симметричное мультипроцессирование и многозадачный режим работы, обеспечивает контроль доступа к информации, а также содержит средства защиты пользователей друг от друга и от сбоев системы.

Windows NT включает два уровня (режима), каждый из которых состоит из двух классов компонентов. Компоненты, исполняемые в привилегированном (системном) режиме, имеют непосредственный доступ к аппаратным ресурсам. Они включают:

- исполнительный модуль, обеспечивающий базовые операции ОС, требуемые для работы остальных компонентов;
- расширения привилегированного режима, включая драйверы устройств, файловую систему и средства управления процессами, требующими тесного взаимодействия с аппаратно-зависимыми компонентами ОС.

Компоненты, исполняемые в непривилегированном, пользовательском режиме, являются равноправными процессами, требующими поддержки средств системного уровня:

- защищенные подсистемы различных пользовательских интерфейсов;
- прикладные программы пользователей.

Исполнительный модуль осуществляет базовые операции управления системными ресурсами, не зависящие от конкретной операционной среды, то есть одновременно обслуживает различные подсистемы, например UNIX и Windows. Он состоит из аппаратно-зависимого ядра (приблизительно 50 Кбайт), обеспечивающе-

Таблица 5

Модель	Операционная система
DECstation	ULTRIX, Windows NT
DECsystem	ULTRIX
Indigo	IRIX 4.0
Crimson	IRIX 4.0
Magnum 3000	RISC/OS,
Magnum 4000	RISC/OS, Windows NT
Millenium 4000	RISC/OS, Windows NT

Продолжение. Начало в КомпьютерПресс № 5'93.

го, кроме всего прочего, синхронизацию мультипроцессорного режима.

32-разрядная несекционированная модель виртуальной памяти с полным разделением и защитой процессов исключает необходимость дополнительных пересылок массивов данных между программой и системными буферами внешних устройств. Таким образом, исполнительный модуль, файловая система и драйверы устройств используют разделяемый пул программных буферов.

Драйверы внешних устройств организованы в виде библиотек, они имеют двухуровневую структуру, причем только часть верхнего уровня привязана к конкретному устройству. Следовательно, для написания нового драйвера обычно необходимо внести минимальные изменения только в эту часть. Надо также отметить, что драйверы могут динамически загружаться в процессе работы.

Файловая система построена по принципу драйвера высокого уровня. Это, в частности, позволяет в рамках одной операционной системы поддерживать различную организацию файлов на внешних устройствах, например DOS-совместимую таблицу расположения файлов (FAT), OS/2-совместимую высокоскоростную файловую систему (HPFS) и файловую систему для CD-ROM.

Windows NT поддерживает три типа интерфейсов прикладных программ, которые могут выполняться одновременно. Новый 32-разрядный (совместимый с 16-разрядным интерфейсом для DOS и Windows), расширенный новыми средствами управления ресурсами и графикой, например семафорами, разделяемой памятью, именованными каналами, сетевыми средствами, кривыми Безье. Этот 32-разрядный интерфейс прикладных программ будет в дальнейшем использован при создании новой ОС Windows32s для микропроцессоров Intel 80x86. Интерфейс в стандарте POSIX обеспечивает возможность исполнения в рамках Windows NT программ для UNIX, а интерфейс OS/2 — программ для OS/2 версий 1.3 и 2.x, например SQL-сервер.

Пользовательские интерфейсы выполнены в виде защищенных подсистем, функционирующих в отдельном адресном пространстве. Для связи с пользовательскими программами реализована процедура удаленного вызова между клиентом — пользовательским приложением и сервером — защищенной подсистемой.

Используемое в Windows NT 16-битное кодирование символов позволяет использовать любую национальную кодировку символов, не только европейские алфавиты, но и иероглифы. Все текстовые системные сообщения собраны в отдельные файлы, что позволяет переводить их на национальные языки, не перекомпилируя объектные модули. Система отвечает требованиям американского стандарта C2 на безопасность компьютерных систем. Напомним, что этому же стандарту удовлетворяет ОС VMS фирмы DEC. Кстати, новую ОС планируется аттестовать по более строгому стандарту B1.

RISC/OS

Операционная система RISC/OS — это специально переработанная для использования на компьютерах фирмы MIPS версия ОС UNIX, базирующаяся на двух важнейших стандартах: V.3 и BSD 4.3. Полная совместимость с версией V.3 достигается соблюдением всех интерфейсов, оговоренных в документе SVID (System V Interface Definition), и использованием ядра V.3. Совместимость с версией BSD 4.3. обеспечивается не только на уровне пользовательского интерфейса, но, самое главное, включением всех системных вызовов, библиотек и файлов заголовков, необходимых для компиляции и выполнения программ.

RISC/OS поддерживает быструю файловую систему BSD, гибкую систему символических ссылок, имена файлов увеличенной длины. Кроме того, были переработаны алгоритмы управления заданиями, виртуальной памятью и файловой системой. Для ликвидации узких мест при вводе-выводе были внесены изменения в алгоритмы кэширования. В результате RISC/OS поддерживает быстрые файловые операции со скоростью до 5 Мбайт/с.

Одним из ключевых моментов повышения производительности является наличие RISCcompiler, специально оптимизированного для использования на микропроцессор Rx000, компилятора языка C, поддерживающего стандарты ANSI и Керниган/Ричи, позволяющего производить компиляцию в любом из выбранных окружений. Конечные пользователи могут также воспользоваться языками программирования Фортран, Паскаль, Модула-2, Кобол, Бейсик и т.д.

RISCcomm/Networking — это компонент ОС, предназначенный для обслуживания различных сетевых и коммуникационных приложений. Он включает также эмуляцию BSD-терминала. В течение одного сеанса работы пользователь может связываться с различными компьютерами, производить обмен информацией, запускать задачи на удаленных ЭВМ. Сетевые средства имеют встроенную поддержку сверхбыстрых протоколов, таких как FDDI и UltraNet.

RISC/OS 4.5i содержит средства для поддержки национальных алфавитов, используя при необходимости 16-битную кодировку символов, национальный формат дат, денежных единиц, последовательность расположения символов в национальном алфавите.

IRIX 4.0

Операционная система IRIX 4.0 фирмы Silicon Grafics основывается на ОС UNIX V.3, в которую включены сетевые расширения BSD 4.3, а также различные сетевые протоколы: TCP/IP, SNA(IBM), DECnet. Изменения, внесенные в ядро системы, позволяют производить в реальном времени обработку аудио- и видеосигналов. Для этой цели разработан специальный набор примитивов мультимедиа — библиотеки стандартных функций для обработки аудио- — Audio Library (AL) и ви-

Таблица 6

Параметры / Модель	SGI Indigo	DECstation 5000/25	HP Apollo 710	IBM RS/6000 7011-220
Процессор	R3000/R3010	R3000/R3010	PA RISC	POWER RISC
Частота, МГц	33	25	50	25
Кэш, Кбайт	2*32	2*64	32+64	2*8
ОЗУ, Мбайт	16	16	32	64
НЖМД, Мбайт	236	426	420	400
Адаптеры	SCSI Ethernet Audio DAT 2*RS232 Centronics	SCSI Ethernet Audio - RS232 -	SCSI Ethernet Audio DAT 2*RS232 Параллельн	SCSI Ethernet - - 2*RS232 Параллельн
Графическая подсистема	1024*768*8	1024*768*8	1280*1024	1024*768*8
Производительность, MIPS	22,14	16,76	20,67	14,09
Drystone (оптим.)	50050	40983	92936	54406
Whetstone (двойн. точн.)	17921	19481	47170	17182
Linpack (двойн. точн.)	3194	2157	8918	4736
X8ench	53964	34504	111181	44062
Стоимость [DM]	24900	20060	47546	24659

деосигналов — ImageVision Library (IL). С помощью первой из них компьютер превращается в 4-канальный микшерский пульт, с возможностью воспроизведения различных эффектов, например "эхо", спектральной цифровой обработки и синтеза сигналов. Вторая библиотека состоит из 70 подпрограмм (методов), позволяющих производить разнообразную обработку видеоизображения, например масштабировать, преобразовывать изображение в файлы в формате TIFF, SCI, FIT.

ОС IRIX 4.0 поставляются со специальным программным модулем графических библиотек (Graphics Library) — набором из более чем 450 функций для работы с трехмерной графикой, которые могут быть вызваны из пользовательских программ, написанных на языках C, C++, Фортран, Паскаль и Ада. В основе лежит описание графического объекта в виде набора точек, контуров и поверхностей, а также таких свойств объекта, как прозрачность и степень отражения света. Графическая библиотека предназначена для обработки изображений в распределенных системах и является аппаратно-независимой. Сегодня GL становятся промышленным стандартом для многих рабочих станций. Такие фирмы, как IBM, DEC, Compaq, Intel, Microsoft, SCO, уже заключили лицензионные соглашения на использование GL в своих разработках.

Пользователь имеет возможность выбирать различные графические интерфейсы. В соответствии с требованиями стандарта X11R4 графические интерфейсы пользователя позволяют производить сложные манипуляции с трехмерными изображениями: вращения, масштабирование. Следует выделить модуль SHOWCASE, позволяющий объединить в одном документе тексты и трехмерную графику. Новый модуль EXPLORER предназначен для моделирования движущихся трехмерных изображений.

Сегодня существует более полутора тысяч приложений, работающих в IRIX 4.0. Многие программные продукты, первоначально разработанные для персональных компьютеров, сейчас перенесены в новую операционную среду.

Сравнение рабочих характеристик компьютеров

Интересно сравнить характеристики новейших рабочих станций, выполненных с учетом спецификации ARC, — DECstation 5000/25 и IRIS Indigo, с имеющимися на рынке конкурирующими моделями — HP Apollo 9000-710 и IBM RS/6000 7011-220. Все сравниваемые модели представляют собой базовые комплексы, выполненные в собственных конструктивах, что исключает возможность установки модулей других производителей. Они имеют сходные технические характеристики. Все рабочие станции, кроме IBM, имеют входы для подключения высококачественных источников аудиосигналов, а в Indigo и HP Apollo предусмотрена возможность подключения устройств типа DAT. Результаты проведенного журналом iX тестирования приведены в табл. 6.

Рабочие станции HP Apollo по-прежнему остаются одними из самых производительных на компьютерном рынке. Так, модель 9000/710 имеет по ряду показателей более высокую производительность, чем остальные модели. Однако применение микропроцессора PA-RISC, работающего на частоте 50 МГц, делает эту модель и в два раза дороже. Модель IBM RS/6000 7011-220 уступает по производительности и IRIS Indigo, и DEC, но цены на эти модели практически одинаковые.

В модели DECstation кэш данных вдвое больше, чем в Indigo, поэтому на коротких тестах (Whetstones) она имеет небольшое преимущество в быстродействии. В то же время на коммерческих тестах (Drystones) полностью проявляется и большая частота работы процессора, и повышенная более чем в два раза пропускная способность шины. В вычислениях с плавающей точкой (Linpack) модель IRIS Indigo оказалась на третьем месте после HP и IBM. По производительности графической системы (XBench) она уступила только HP. По общесистемной производительности Indigo имеет очень близкие показатели с HP и в 1,5 раза превосходит DECstation 5000/25.

Рабочие станции DEC работают на более низкой частоте, чем IRIS Indigo, и отличаются меньшей про-

пускной способностью системной шины. С учетом того, что стоимость элементов значительно возрастает с увеличением быстродействия, разница в цене между моделями DEC и SGI представляется незначительной. К этому необходимо добавить, что Indigo обладает развитыми графическими возможностями и более совершенными средствами мультимедиа, ориентированными на обработку сигналов в реальном времени. Таким образом, можно говорить о том, что по показателю производительность/стоимость модель Indigo превосходит DECstation 5000/25.

При сравнении предлагаемых операционных систем следует обратить внимание на те дополнения, которыми обладает Windows NT. Например, высокий уровень защиты информации, поддержка модели распределенной обработки, большая гибкость в работе и удобный прикладной интерфейс способны привлечь большой интерес пользователей. Пока еще слишком рано говорить, какое число пользователей привлечет Windows NT. Сегодня в этом секторе доминирует UNIX. Известно, что в ближайшее время и RISC/OS, и IRIX 4.0 будут доработаны с учетом дополнений последней версии UNIX 5.4. В обеих ОС отдается предпочтение графическому интерфейсу OSF/Motif.

Описываемые модели компьютеров, созданных со спецификации ARC, были представлены на выставке CeBit'92. И если главной новостью 1991 года явилось известие об объединении двух крупнейших германских компаний Siemens и Nixdorf, то в 1992 году таким известием стало объявление о соглашении между фирмами Silicon Grafics и MIPS Computer Systems об объединении усилий в области создания и разработки новых поколений средств вычислительной техники. Этот альянс позволит обеим фирмам стандартизовать используемые технические решения, объединить достижения фирмы SGI в области компьютерной графики с достижениями фирмы MIPS в создании серверов для компьютерных сетей. Было также заявлено о подписании долгосрочных соглашений между Silicon Grafics и Siemens Nixdorf о совместных разработках в области рабочих станций с архитектурой ARC. Отличительной чертой разработок фирмы Siemens Nixdorf является комплексный подход к решаемой проблеме, интеграция новых технических решений с существующими. Можно ожидать, что современные компьютеры с архитектурой ARC будут легко объединяться с существующими вычислительными комплексами, выгодно дополняя их высокой вычислительной мощностью и отличными графическими возможностями. Учитывая тот факт, что Siemens Nixdorf уже приступила к реализации своих широкомасштабных проектов на отечественном рынке, следует ожидать скорого появления новейшей техники в стандарте ARC и в нашей стране.

Инициатива ACE не избежала болезней роста. Уповая на будущий успех широко рекламируемого RISC "Проекта Альфа", фирма DEC объявила о выходе из консорциума, сохраняя, однако, ЭВМ серии 5000 в производственной программе. Но несмотря на это, инициатива имеет высокие шансы на успех, поскольку ее участники первыми выпустили на рынок высококачественные ЭВМ нового поколения.

И.Федоров, А.Гиглавый

В статье использованы материалы зарубежной компьютерной прессы.

Лазерные принтеры Kyocera

Семейство лазерных принтеров фирмы Kyocera включает несколько моделей: F-800, F-1000, F-1200S, F-1800, F-220S и F-3000. Все эти модели различаются, разумеется, не только по весу и габаритным размерам, но и по вычислительной мощности используемых процессоров, производительности и уровню предоставляемого сервиса. В этой небольшой статье мы постараемся коротко остановиться на некоторых характеристиках представляемых моделей. Все другие интересующие вас вопросы вы можете задать сотрудникам фирмы B&K, поскольку именно она начинает поставку этой техники в Россию.

С модели F-800 фирма Kyocera начала серию своих новых лазерных принтеров, которые отличает малый вес, небольшие габаритные размеры, высокая скорость печати при приемлемом качестве и простота обслуживания и управления. Для начала постараемся выделить наиболее общие черты принтеров Kyocera. По понятным причинам модель F-800 в ряде случаев может несколько отличаться от остальных.

Все модели серии F имеют максимальную разрешающую способность 300x300 точек на дюйм. Для всех моделей может использоваться графический язык описания страниц PRESCRIBE. Любая модель может эмулировать работу таких принтеров, как HP LaserJet II, IBM Grafikdrucker (графический принтер), Diablo 360, Qume Sprint 11, NEC Spinwriter, Line Printer, а также Epson FX-80. Принтеры Kyocera могут использовать до 78 шрифтов, "зашитых" в ПЗУ, и 3 шрифта с международными символами. Размер такого ПЗУ составляет не менее 1 Мбайта. Для загружаемых (download) шрифтов возможно применение формата как Kyocera, так и HP LaserJet II. Кстати, наличие двух слотов позволяет подключать к принтерам устройства типа IC Card (размером с кредитную карточку и внешне очень напоминающее модуль флэш-памяти), на которых можно хранить не только загружаемые шрифты, но и другую необходимую информацию (например, логотипы фирм, макросы). Подача бумаги в принтер может осуществляться как из кассеты, так и отдельными листами. Высокое качество печати обеспечивается данными принтерами в немалой степени благодаря двухкомпонентному тонеру. Кстати, расход тонера в расчете на одну кассету составляет около трех—четырех тысяч листов.

Уровень шума, создаваемый работающим принтером, обычно не превосходит *55* дб. Принтеры Kyocera могут подключаться к компьютерам не только через параллельный, но и через последовательный интерфейс.

Все рассматриваемые модели принтеров основаны на 16- или 32-разрядном микропроцессоре фирмы Motorola, соответственно MC68000 и MC68020. Скорость печати принтеров серии F варьируется в зависимости от модели и составляет от 8 до 18 страниц в минуту. Как известно, принтеры подобного класса печатают со скоростью около 6 страниц в минуту. Кассеты для бумаги вмещают от 150 до 250 форматных листов, а модели F-2200S и F-3000 оснащены двумя такими кассетами. Практически все модели принтеров снабжены LCD-экраном, на котором, в частности, могут отображаться указанные режимы работы, выбранная кассета с бумагой и устройство складирования. Оперативная память принтеров легко наращивается и составляет практически для всех моделей не менее 1 Мбайта.

Модели / Параметры	F-800	F-1000	F-1200S	F-1800	F-2200S	F-3000
Тип процессора	MC68000	MC68000	MC68020	MC68020	MC68020	MC68000
ОЗУ, Мбайт	0,5;1;2;4	0,5;1	1;2;3;4	1;2;3;4	1;2;4	1,5;2
ВидеоОЗУ, Мбайт	–	–	+	+	+	2
ПЗУ шрифтов, Мбайт	1,5	1	1	1,5	1,5	1
Скорость печати, страниц/мин	8	10	10	18	10	18
Разрешение, точек/дюйм	300x300	300x300	300x300	300x300	300x300	300x300
Количество листов в кассете	250	250	250	250	2*250	2*250
Расход тонера, листов/кассета	4000	3000	3000	4000	3000	3000
Размеры, мм	406x390 x425	428x320 x450	428x320 x450	428x330 x450	428x370 x450	428x370 x450
Вес, кг	17	26	26	26	35,5	35,5

Модель F-800 доступна пользователям в трех вариантах: F-800T — для работы в основном с текстовой информацией, F-800TI — для работы с изображениями (например, в составе небольших настольных издательских си-

стем) и F-800TIV — для работы с векторной графикой (представляет интерес для специалистов в области САПР). Для приложений САПР имеется и более производительная модель — F-1200S. Но настоящие профессионалы не смогут остаться равнодушными к таким высокопроизводительным моделям, как F-1800 и F-3000, скорость печати которых составляет 18 листов в минуту.

Кстати, для наращивания производительности и удобства работы с принтерами серии F фирмой Kyocera предлагаются, например, модули расширения памяти, устройство для программирования IC Card, устройство последовательной подачи бумаги, пятиуровневый сортировщик и программное обеспечение для создания и корректировки шрифтов. В частности, пятиуровневый сортировщик позволяет разделять информацию, предназначаемую для каждого (из 5) конкретного пользователя. Устройство последовательной подачи бумаги PF-1 автоматизирует загрузку до 1000 форматных листов, что особенно важно при работе с такой высокопроизводительной моделью, как F-3000.

Итак, новые производительные принтеры Kyocera — на нашем рынке. На все возникшие вопросы вам ответят по телефону фирмы B&K: (095) 110-47-63.

А.Борзенко

Фирма Dell входит в десятку крупнейших производителей и поставщиков персональных компьютеров. Компьютеры с маркой Dell появились недавно и на российском рынке. Сегодня мы расскажем о новой модели ноутбука NL25.

Компьютер Dell NL25

Первые впечатления

В отличие от других моделей компьютеров-блокнотов, тестируемых нашей редакцией, ноутбук Dell NL25 был предоставлен для исследования нашим же коллегой — И.Могучевым, которому мы и выражаем свою благодарность. Поскольку ноутбук был приобретен непосредственно в США, то автора этих строк, естественно, интересовали подробности того, как продают компьютеры-блокноты "у них". Хотелось бы сразу отметить "ненавязчивый" американский сервис. Например, установка встраиваемого факс-модема обходится чуть ли не в его полную стоимость, зато к компьютеру прилагается обилие аксессуаров: описаний, дискет и т.п.

Представленная модель ноутбука основана на экономичном микропроцессоре i386SL-25 и в базовой конфигурации содержит 4 Мбайта оперативной памяти (80 нс), 80-Мбайтный винчестер со временем доступа около 15 мс, а также встроенный привод флоппи-дисков размером 3,5 дюйма. Вес ноутбука вместе со сменным аккумулятором не превышает 2,86 кг (6,3 фунта), габаритные размеры — 11 на 8,3 на 1,8 дюйма (28,0x21,1x5,8 см). "Бумажно-белый" экран дисплея имеет максимальную разрешающую способность 640 на 480 точек и может воспроизводить до 16 оттенков серого цвета. Через два последовательных и один параллельный порт можно подключать к компьютеру соответствующие устройства. В случае установки встроенного факс-модема он занимает адресное пространство одного из последовательных портов. Для подключения внешней полноразмерной клавиатуры и аналогового CRT-дисплея предусмотрены специальные разъемы. Кроме того, ноутбук комплектуется мини-трекболом фирмы Microsoft.

Клавиатура

Модель Dell NL25 имеет 85-клавишную клавиатуру с возможностью эмуляции клавиш обычной 101-клавишной клавиатуры. Одновременное нажатие служебной клавиши Fn и одной из функциональных клавиш F1-F12 используется для специальных функций. Например, для изменения тактовой частоты работы процессора необязательно вызывать программу Setup, так как это можно сделать с помощью комбинации клавиш Fn-F3 (минимальная), Fn-F4 (максимальная) на встроенной клавиатуре. Наличие выделенных клавиш F11 и F12, как известно, дает в некоторых случаях определенные преимущества.

На клавиатуре имеется по две клавиши Shift и Alt, но только одна клавиша Ctrl, что может создать определенные неудобства, если обычно вы используете программу-русификатор клавиатуры с "горячей" клавишей "правый" Ctrl. Клавиша Caps Lock расположена в нижнем ряду клавиатуры рядом с клавишей Fn. Полноразмерные клавиши управления курсором расположены в форме "перевернутое T", что, как известно, очень удобно. Двенадцать функциональных клавиш (F1-F12) находятся на привычном месте — в верхней части клавиатуры. Что же касается клавиш, интенсивно используемых, например, при редактировании текстов, то они расположены "столбиком" в правой части клавиатуры: Home, PgUp, PgDn, End. Режим встроенной цифровой клавиатуры (размещенной, разумеется, в поле алфавитно-цифровых клавиш) доступен после нажатия выделенной клавиши NumLock.

Дисплей

Ноутбук Dell NL25 оснащен FTN VGA-дисплеем с задней подсветкой экрана. В режиме REVERSE цвет экрана практически "бумажный". При разрешающей способности 640 на 480 точек такой LCD-экран может воспроизводить до 16 оттенков серого цвета. Видеоконтроллер фирмы Cirrus Logic оснащен 256 Кбайтами памяти, что позволяет поддерживать воспроизведение 17 стандартных VGA-режимов, в том числе и полностью совместимые с MDA, CGA и EGA.

На экране размером чуть больше 24 см (9,5 дюйма) в 16 оттенках серого цвета может эмулироваться 16- или 256-цветная палитра.

Как известно, LCD-экран с подсветкой является одним из самых "прожорливых" элементов компьютера-блокнота. Использование программы Power Management

Setup (PM Setup) позволяет использовать экономичные режимы работы. Например, если вы не работаете с мышью или клавиатурой в течение некоторого промежутка времени, называемого "таймаут", может происходить гашение LCD-экрана (режим Standby) что, соответственно, уменьшает общее энергопотребление компьютера. Значение параметра "таймаута" может изменяться в пределах от 0 до 17 минут.

Модель Dell NL25 оснащена 2,5-дюймовым винчестером емкостью 80 Мбайт, причем среднее время доступа составляет около 15 мс, а скорость передачи информации — примерно 670 Кбайт/с. Данный винчестер, как обычно, имеет достаточно производительный системный интерфейс IDE.

Напомним, что для сохранения энергии аккумуляторов винчестер может отключаться, если к нему нет обращения в течение времени, указанного в программе PM Setup. Этот интервал может составлять от 0 секунд до 17 минут, после чего винчестер переходит в состояние Standby.

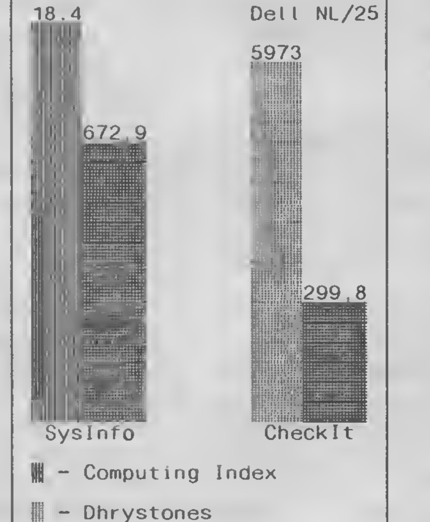
Встроенный привод флоппи-дисков размером 3,5 дюйма позволяет работать не только с дискетами емкостью 1,44 Мбайта, но и поддерживать ряд других форматов.

Работа в автономном режиме

Ноутбук модели Dell NL25 наиболее эффективно может использоваться именно в автономном режиме по двум причинам. Во-первых, базовый микропроцессор i386SL-25 имеет все необходимые встроенные особенности для организации экономичных режимов работы, и, во-вторых, фирмой Dell полностью поддержана спецификация Intel/Microsoft APM (Advanced Power Management). Функции APM реализует специальный программный драйвер. Как известно, поддержка APM требует использования MS-DOS 5.0 или Windows 3.1.

Среднее время работы ноутбука от никель-кадмиевого аккумулятора емкостью 24,5 Втхч составляет около 2,5 часов. Отметим, что весит аккумулятор всего около 0,68 кг (1,5 фунта). Подзарядка аккумулятора выполняется от AC-адаптера в течение 2-3 часов, при работе с ноутбуком это время увеличивается до 9-10 часов.

С помощью программы PM Setup для ряда устройств (дисплей, винчестер, последовательный порт, модем) можно установить временные интервалы, по истечении которых данные устройства, если к ним нет обращения, переходят в режим Standby. Кстати, этот режим можно ввести и вручную, используя комбинацию клавиш Fn-F5.



Как известно, другим, более "продвинутым", экономичным режимом, который доступен через программу PM Setup, является режим Suspend. Этот режим позволяет корректно прерывать на время выполнение текущей программы. При этом, как правило, происходит отключение наиболее энергозависимых устройств и закрытие системы с сохранением текущего статуса. Работа приостановленной программы может быть продолжена с того же места, где она была прервана. Обычно режим Suspend может вводиться как вручную (например, после нажатия комбинации "горячих" клавиш или закрытия дисплея ноутбука), так и автоматически по истечении заданного времени таймаута. Ввод режима Resume выполняется нажатием комбинации тех же "горячих" клавиш или после того как открыт дисплей.

Программа PM Setup в модели NL25 позволяет, например, автоматически вводить (интервал от 1 до 30 минут) и выводить компьютер из режима Suspend, соответственно режимы Autosuspend и Autoalarm. К тому же в вышеназванной программе для процессора можно выбрать одну из тактовых частот — 25 или 6 МГц.

Возможности расширения

Немаловажным достоинством ноутбука Dell NL25 является возможность установки математического сопроцессора i387SL/SX. Это особенно актуально, если используемые вами программы связаны с интенсивными вычислениями. Оперативная память ноутбука может быть легко расширена до 12 Мбайт путем замены модулей памяти.

Дисковая память компьютера наращивается заменой 80-Мбайтного винчестера на 120-Мбайтный. Для подключения 101-клавишной клавиатуры и CRT VGA-дисплея на корпусе ноутбука имеются специальные разъемы. Для принтера на корпусе ноутбука предусмотрен разъем параллельного порта, который, кстати, может поддерживать и внешний привод флоппи-дисков. Модуль встроенного факс-модема (2400/9600 бит/с) поддерживает протоколы MNP 5/V42/V42bis, причем факс может использоваться как для приема, так и для передачи. При работе с Windows незаменимым инструментом является внешний трекбол фирмы Microsoft (Ball Point Mouse).

Программное обеспечение и документация

В качестве базовой операционной системы для ноутбука Dell NL25 используется MS-DOS версии 5.0, кото-

при покупке компьютера. К этим двум программным продуктам придаются весьма объемные руководства пользователя. На двух 3,5-дюймовых дискетах (Diagnostics и Software Support), входящих в комплект ноутбука, записан ряд специальных системных драйверов и утилит, облегчающих работу и диагностику неисправностей. На них имеются соответствующие описания.

Документация по самому ноутбуку включает брошюры Getting Started и User's Guide. Хотя они и не очень велики по объему, но достаточно подробны. В них можно найти ответы на самые разнообразные вопросы, вплоть до описания сигналов на контактах используемых портов. Безусловно, основное внимание уделяется аспектам именно практической работы на компьютере.

В комплект дополнительного модуля факс-модема включены средства программной поддержки — пакет Quick Link II (на 3,5-дюймовой дискете), снабженный подробным описанием. Имеется описание и на сам модуль факс-модема. В комплект с трекболом также входят 3,5-дюймовая дискета и описание.

Фирма Dell гарантирует безотказную работу своего изделия в течение одного года, на это время пользователи обеспечиваются необходимым сервисом и технической поддержкой.

А.Борзенко

Сторонний взгляд на МАСТЕРа и "мастеровых"

— "Мастер" — это такой "Лексикон"?
— Нет, это такой "Фреймворк"...

Вначале было Слово. Слово было одно. Слово было загадочное — "Мастер"...

Вслед за статьей Е.Н.Веселова о средо-ориентированном программировании (в самом первом номере самого первого отечественного компьютерного журнала — PC World-USSR), где ИС МАСТЕР только упоминалась, появилась и книга: "Интегрированная система МАСТЕР для ПЭВМ". Потом вдруг выяснился интересный факт, что МАСТЕР известен не меньше, чем МММ, а проблем у него не больше, и масса довольных пользователей пишет, а также считает и рисует только по-русски — в МАСТЕРе и в МАСТЕР-приложениях, созданных не менее довольными МАСТЕР-программистами. Ну, а инженеры СП МИКРОИНФОРМ, работающие под руководством Веселова, трудятся над дальнейшим развитием МАСТЕР-интерфейса...

Все довольны? Да нет, сам же Веселов и недоволен. Framework, Paradox, Clipper и МАСТЕР 1 — вчерашний день, пора создать, наконец, среду истинно визуального (термин "средо-ориентированное" уходит на покой?) программирования! МАСТЕР 2 станет открытой, переносимой между разными операционными средами и платформами, системой.

Дело это новое, и, несомненно, потребуется бета-тестирование. Испытание новых продуктов "на живых людях" — едва ли не важнейшая часть подготовки программного обеспечения к распространению. И вот мы сидим в уютном классе, под знаменем Novell Education. На наших глазах идет потрясающий процесс созидания Нового.

В фундаменте постройки нового МАСТЕРа лежат язык С и объектно-ориентированное программирование. Но не "приглянулись" создателям МАСТЕРа для их (как убедимся ниже — весьма и весьма благородных) целей ни С++, ни Объектный С, и принципы ООП реализованы ими в рамках процедурного стандарта. Таким образом, основа Визуального Мастера, то есть библиотеки классов объектов, процедур и функций, заложена в рамках "обычного" С с его простой и ясной семантикой, не требующей переучивания программистов. И оттуда же, из С, открыт доступ для создания новых базовых модулей и пакетов.

Так где же революционные новшества? В первую очередь — в системе организации памяти. Чудо, сотворенное Олегом Князьковым, называется Virtual Memory Manager. Это не просто еще одна система управления виртуальной памятью, и умеет она не только "расширять" ОЗУ до объема жесткого диска, прозрачно поддерживая при этом XMS и EMS. Разработчики пошли дальше и создали систему программирования, не зависящую от среды DOS или Windows! VMM поддерживает тщательно согласованную с требованиями среды Windows технологию динамически компонуемых библиотек DLL. Созданные МАСТЕРом DLL-модули можно просто "подбрасывать" Windows, а для работы под DOS система VMM сама обеспечит их динамическую загрузку и связывание.

Но в конечном-то счете, для чего весь этот сыр-бор? Конечно, для создания полноценного событийно-управляемого оконного интерфейса! Семинар по пакету CUI — Common User Interface — провел автор пакета Евгений Дмитриев. Все меню, окошки, кнопочки и прочие элементы пользовательской среды, создаваемой средствами нового МАСТЕРа, являются объектами, опирающимися на мощную VMM-поддержку, и характеризуются — помимо структуры и внешнего вида — определенным поведением.

Таков базис, над которым возведена система визуального программирования. Мы наблюдаем за тем, как совершенствуется Визуальный Конструктор, созданный именно для того, чтобы стало возможным строить информационную среду изнутри — и не для ИС МАСТЕР, как раньше, а для DOS и Windows! Таким образом, можно очень быстро разрабатывать основные интерфейсы будущей прикладной системы. Этот процесс — настоящий клад как для неподготовленного пользователя, так и для измученного рутинной работой программиста — "веселовцы" зовут "визуальной раскруткой".

Но собственно информационная среда — это всего лишь скелет. Всем этим кнопочкам, переключателям и меню еще надо придать смысл. И автор языка МАСТЕР читает нам лекции по новейшему диалекту, реализованному в системе МАСТЕР 2. На встроенном языке программируются всевозможные аспекты поведения прикладной программы — от функций кнопок до организации баз данных. ЛЕКСИКОН 2 теперь сделан "на чистом МАСТЕРе" — и благодаря этому он может быть интегрирован с другими частями МАСТЕР-технологии. Отныне вы получаете возможность "прикупить" к вашему ЛЕКСИКОНу (или к любому другому программному продукту, созданному на МАСТЕРе), например, графический редактор или электронную таблицу для того, чтобы работать с ними в общей среде и не тратиться на приобретение независимых программных изделий.

Когда бы ни появлялось в мире что-нибудь новое (колесо, ЭВМ, ООП...), всегда было полным-полно людей, снобистски заявлявших, что они "отлично могут по старинке" (ездить на полозьях, считать на счетах, программировать на Фортране...). Вот вам, например, никогда не хотелось спросить у Веселова: "А зачем вы все это, господин хороший, делаете? Вон сколько есть на свете прекрасных текстовых процессоров, интегрированных пользовательских систем, мощных языков программирования!" Как человек деликатный и мягкий, он на такой вопрос ответит: "Если принять вашу точку зрения, то нам всем останется лишь дружно взять и застрелиться..."

А "мастеровые" предпочитают жить — и МАСТЕРски работать.

К.Ахметов



Тятя, тятя, наши сети...

Кому достанется богатый улов компьютерных сетей?

Еще несколько лет назад компьютерные сети были уделом ученых и фанатичных любителей, о серьезных прибылях в этой области говорить не приходилось. Крупные системы, мощные вычислительные ресурсы, специализированные высокоскоростные каналы связи — даже в опережавшей всех Америке доступ к ним давали либо правительство руками военных (которые денег не считали), либо корпорации, выполнявшие научно-исследовательские работы для тех же военных.

Сегодня ситуация стремительно меняется — Европу, Северную Америку и Японию захватывает процесс быстрой коммерциализации систем компьютерной связи, образуются альянсы и выбираются союзники. Буквально на глазах формируются международные консорциумы, стремящиеся занять на рынке нишу покрупнее.

Многочисленные эксперты и аналитики утверждают, что экономические события 90-х годов в значительной мере будут определяться развитием информационного рынка, связанным с повальным объединением миллионов машин в глобальные компьютерные сети, бумом, аналогичным тому, который был вызван появлением персональных компьютеров в 80-х годах, но превосходящим его по масштабам.

С одной стороны, новейшие технологии, возникающие на стыке электросвязи и компьютеров, обеспечивают невиданные возможности глобальной связи, что открывает широкий путь такому товару, как информация. С другой — большое число традиционных отраслей рассматривает эти компьютерные сети как мощный и эффективный канал сбыта. Телекоммуникации, издательская деятельность, телевидение неизбежно превратятся в одну гигантскую индустрию, предоставляющую все возможные информационные услуги. Эту тенденцию можно было заметить по ряду совместных проектов газетных концернов с компьютерными сетями, по тому, как рвутся на информационный рынок западные телефонные компании, а совместный проект компьютерного монстра IBM и Time/Warner — компании номер 1 среди средств массовой информации и киновидеостудий США — стал заметным сигналом даже для тех, кто далек от последних технологических достижений. Голос, данные, изображения, видео — все это пойдет через один и тот же канал: компьютер плюс линия связи.

Образование этой индустрии будет, как всегда, сопровождаться сверхприбылями, их получат те, кто успел прыгнуть в отходящий поезд и перегнал соперника. А побороться есть за что: размер рынка информационных технологий, не включая собственно технические средства связи, уже в настоящее время перевалил за 3 триллиона долларов в год и продолжает быстро расти.

Части этого рынка уже осваиваются. Американская компьютерная мегасеть Internet с 1989 года растет со скоростью 20 процентов в месяц и уже насчитывает более миллиона подключенных компьютеров. Число пользователей таких японских сетей персональных компьютеров, как PC-Van и Nifty-Serve, в 1992 году увеличилось на 80 процентов. Не отстают и европейцы. Строительство сетей и их проектирование идут во всем мире практически одновременно, порождая массу конфликтов и противоречий.

Ситуация осложняется тем, что в современных условиях любая сеть связи становится компьютерной. Поэтому на образующийся рынок стремятся попасть телефонные компании и компании кабельного телевидения. Одним из предвестников надвигающейся битвы является "конец удельных княжеств и уютных монополий" государственных телефонных сетей — демонополизация телефонных услуг в Великобритании и Австралии, недавнее объявление о приватизации Deutsche Bundespost, активно идущая распродажа частным вла-

дельцам государственных систем связи в Азии и Латинской Америке. Европейская Комиссия заняла чрезвычайно жесткую позицию в вопросе о взаимном открытии телекоммуникационного рынка стран—членов ЕС — нужно успеть вырастить соперников для американских гигантов AT&T и MCI.

Таким образом, цель битвы ясна — ключевые позиции в борьбе за прибыли и сверхприбыли в формирующемся на глазах информационном обществе. В постиндустриальном мире победители этой битвы займут то место, которое век назад занимала тяжелая промышленность. Известны и участники — транснациональные компании, обеспечивающие техническую и информационную поддержку глобальных информационных систем. Однако элемент своеобразия в диспозицию вносит явное несоответствие грандиозности масштабов и сравнительной скромности ресурсов участников. Ситуация обостряется тем, что подобные рынки характеризуются большим числом ниш, заполняемых мелкими и средними компаниями. Эти факторы провоцирует постоянный процесс создания группировок и соглашений о разделе сфер влияния.

Компании разных стран объединяются для предоставления действительно глобального сервиса. Примером этого является Commercial Internet Exchange (CIX) — объединение средних и мелких, а впоследствии и крупных фирм США и Европы, предоставляющих услуги по передаче сообщений и информационный сервис. В это объединение входят такие компании, как Sprint, PSI, UUnet Technologies, пан-европейский сетевой консорциум EUnet и другие. Кстати, влияние CIX уже распространилось на нашу страну — в эту ассоциацию входит и крупнейшая компьютерная сеть EUnet/Relcom.

Не стоят в стороне и региональные компании, которые объединяются для вторжения на внешние рынки и защиты внутренних. В Японии прошлой осенью образовалась Японская Ассоциация Электронной Почты, несмотря на то, что входящие в нее наряду с другими NEC и Fujitsu являются прямыми конкурентами и сотрудничают с соперничающими американскими компаниями. Даже AT&T, утверждающая, что ей принадлежит 40 процентов от общего объема систем электронной почты во всем мире (эта цифра явно завышена), идет на соглашения о взаимной проницаемости, например, с английской British Telecom.

А теперь самая важная деталь. Похоже, что на этот раз светлое будущее не обойдется без, казалось, навсегда забытых серьезным рынком стран третьего мира и бывшего "развитого социализма".

Одним из важных потребительских качеств компьютерной сети является "дальность охвата" — то есть возможность передачи информации в любую точку земного шара. Отсутствие больших регионов в зоне своего влияния может оказаться фатальным для участников битвы за сетевые рынки. Это делает важным для европейских и американских компаний включение в свою инфраструктуру телекоммуникационных сетей в странах третьего мира и бывшего соцлагеря, и прежде всего в странах, ранее входивших в СССР, а также Юго-Восточной Азии.

В связи с этим развитие компьютерных сетей в нашей стране уже сейчас определяется не только внутренними причинами, но и внешним влиянием. Можно сделать вывод не только о том, что грядет неизбежный бум телекоммуникационной индустрии, но и о том, что мы сможем наблюдать его, не покидая родных пределов, причем общее состояние экономики не будет столь бесцеремонным диктатором, как в сырьевых отраслях. Если же внимательно приглядеться к готовящимся к битве соперникам, то можно будет получить и финансовую выгоду от начинающегося сражения.

Е.Пескин, В.Бардин



Опасная безопасность

О возможном подходе к решению проблемы выбора оптимальной концепции защиты информации в компьютерных системах...

В настоящее время, несмотря на большое количество проведенных исследований, единая и общепринятая теория безопасности информации в компьютерных системах еще не создана [1,2]. По этой причине существующие подходы, методы и средства обладают рядом видимых и невидимых недостатков, снижающих в итоге ожидаемую эффективность защиты информации от НСД [1].

Надеюсь, приведенные ниже результаты исследований в какой-то мере помогут решить эту проблему.

Подход к решению проблемы

Подход заключается в разработке общей теории защиты какого-либо предмета от несанкционированного доступа и приложении ее к компьютерным системам с уточнением предмета и объекта защиты.

Общая теория защиты

Рассмотрим модель простейшей системы защиты какого-либо предмета, состоящей из одной однородной преграды, замыкающейся вокруг него. Прочность такой защиты зависит от способности преграды противостоять попыткам ее преодоления нарушителем. Свойство предмета защиты — способность привлекать нарушителя. Соотношение свойств предмета и системы защиты определяет эффективность защиты. Известно, что это соотношение может выражаться в отношении ожидаемых финансовых затрат нарушителя на преодоление преграды и стоимости предмета защиты. Если последняя меньше затрат нарушителя, уровень эффективности защиты считается достаточным.

Возможен также выбор критерия оценки по соотношению затрат времени на преодоление преграды нарушителем и времени жизни предмета защиты.

Известно, что информация со временем может терять свою ценность. Тогда условием достаточной прочности преграды при отсутствии путей ее обхода можно считать превышение затрат времени нарушителя над временем жизни информации. Это соотношение можно представить в виде простой формулы:

$$P_{сзн} = (1-P_{нр})(1-P_{обх}) \qquad (1)$$

где $P_{сзи}$ — вероятность непреодоления преграды нарушителем или прочность преграды; $P_{нр}$ — вероятность преодоления преграды нарушителем за время, меньшее времени жизни информации; $P_{обх}$ — вероятность обхода преграды нарушителем.

Условие достаточности выражается следующим соотношением: $P_{сзн}=1$, если $t_ж<t_н$ и $P_{обх}=0$. Выражение $(1-P_{обх})$ определяет степень замыкания (охвата) преградой предмета защиты. При $P_{обх}=1$ защита не имеет смысла.

В качестве примера защиты, рассчитываемой по формуле (1), может быть названо криптографическое преобразование информации, где величина $P_{нр}$ может определяться путем оценки вероятности подбора ключа дешифрования. При этом величина $P_{обх}$ будет зависеть от выбранного метода преобразования, способа применения, хранения кода ключа и т.д., причем возможно также, что у одной преграды может быть несколько путей обхода. В этом случае формула (1) примет вид

$$P_{сзн} = (1-P_{нр})(1-P_{обх1})(1-P_{обх2})\dots(1-P_{обхn}) \qquad (2)$$

где n — число возможных путей обхода преграды.

В случае когда предмет защиты сохраняет свою стоимость и обеспечить $t_н>t_ж$ по каким-либо причинам невозможно (например, информация в компьютерной системе периодически обновляется), применяется преграда, обладающая свойствами контроля и обнаружения. Такой преградой может служить человек или автоматизированная система, периодически контролирующая периметр объекта защиты. Вероятность преодоления преграды нарушителем в этом случае можно выразить формулой

$$P_{нр} = 1 - \frac{t_н}{T_к} \qquad (3)$$

где $t_н$ — время, необходимое нарушителю на преодоление преграды; $T_к$ — периодичность контроля.

Из формулы (3) видно, что при увеличении $t_н$ вероятность преодоления преграды уменьшается, и при $t_н > T_к$ несанкционированный доступ через преграду не имеет смысла — нарушитель будет искать пути ее обхода.

С учетом вероятности возможного отказа ($P_{отк}$) системы контроля выражение для прочности преграды с контролем НСД примет вид

$$P_{сзик} = \frac{t_н}{T_к} (1-P_{отк})(1-P_{обх1})(1-P_{обх2})\ldots(1-P_{обхn}) \quad (4)$$

На практике в большинстве случаев защитный контур состоит из нескольких соединенных между собой преград. Известно, что при попытке преодолеть защиту нарушитель использует наиболее слабую преграду. По этой причине итоговая прочность такой защиты будет определяться прочностью наиболее слабого звена.

Если прочность слабого звена защиты не удовлетворяет заданным требованиям, оно заменяется на более прочное или дублируется еще одной, двумя и более преградами. Тогда итоговая прочность дублированных преград определяется согласно (3) по формуле

$$P_{\Sigma} = 1 - (1-P_{сзи1})(1-P_{сзи2})\ldots(1-P_{сзиN}) \quad (5)$$

где N — порядковый номер преграды.

Предмет и объекты защиты

Предмет защиты — информация, циркулирующая и хранимая в компьютерных системах в виде данных, команд, сообщений и т.д., имеющих какую-либо ценность для их владельца и потенциального нарушителя.

С позиций оптимальной организации и реализации системы безопасности информации все компьютерные системы рассматриваются как объекты защиты, которые делятся на два вида:

- территориально-сосредоточенные системы (комплексы средств автоматизации с централизованной обработкой данных и персональные компьютеры);
- территориально-рассредоточенные системы (вычислительные сети и АСУ).

Определение термина "безопасность информации"

Безопасность информации — это создание таких условий хранения, обработки и передачи информации, при которых вероятность событий ее утечки, модификации или разрушения удовлетворяет заданным требованиям. Такие события, как несанкционированное наблюдение потоков сообщений и навязывание ложной информации, есть частные случаи, соответствующие утечке и модификации информации.

Потенциальные угрозы безопасности информации

Все потенциальные угрозы безопасности информации в компьютерных системах делятся на случайные и преднамеренные. Так как их природа, время и место при-



ложения различны, соответствующие методы и средства защиты также должны отличаться.

Причины случайных воздействий, методы и средства защиты от них известны [4,5]. Время и место их возникновения подчиняются законам случайных процессов. Возможные точки их приложения распределены по всей "площади" системы. Средства защиты от случайных воздействий включают средства предупреждения, обнаружения и блокировки этих событий.

Точки приложения преднамеренных воздействий связаны прежде всего со входами и выходами компьютерной системы, то есть с ее периметром. Эти входы и выходы могут быть штатными и нештатными каналами доступа к информации. К штатным каналам относятся: терминалы пользователей, средства отображения и документирования информации, носители информации, средства загрузки программного обеспечения, внешние каналы связи.

Возможными каналами преднамеренного несанкционированного доступа к информации при отсутствии защиты в компьютерной системе могут быть:

- все перечисленные выше штатные средства при их незаконном использовании;
- технологические пульты и органы управления;
- внутренний монтаж аппаратуры;
- линии связи между аппаратными средствами;
- побочное электромагнитное излучение, несущее информацию;
- побочные наводки на цепях электропитания, заземления аппаратуры, вспомогательных и посторонних коммуникациях, размещенных вблизи компьютерной системы.

Поскольку время и место проявления преднамеренных НСД предсказать невозможно, целесообразно вначале задаться некоторой моделью поведения потенциального нарушителя, предполагая заранее наиболее опасную ситуацию:

1) нарушитель может появиться в любое время и в любом месте периметра компьютерной системы;
2) квалификация и осведомленность нарушителя может быть на уровне разработчика данной системы;
3) постоянно хранимая информация о принципах работы системы, включая секретную, нарушителю известна;
4) для достижения своей цели он выберет наиболее слабое звено в защите;
5) нарушителем может быть не только постороннее лицо, но и законный пользователь системы.

Для компьютерных систем с менее высокими требованиями эта модель может быть изменена и даже нормирована по уровням требований.

Стратегия и тактика защиты информации от НСД

Стратегия и тактика защиты заключается в предупреждении, контроле, своевременном обнаружении и блокировке НСД. При этом на каналах НСД, выходящих за пределы физических возможностей контроля событий, осуществляется только предупреждение НСД, выражающееся в создании защитной преграды, соответствующей времени жизни информации, подлежащей защите.

Реализация стратегии и тактики защиты информации в компьютерных системах заключается в системном подходе и решении следующих задач на этапах проектирования и эксплуатации:

а) на этапе проектирования:

- определение перечня и стоимости данных, подлежащих защите;
- анализ системы как объекта защиты и определение в ней в соответствии с заданной моделью поведения потенциального нарушителя максимально возможного количества каналов НСД к информации и возможных воздействий случайного характера;
- разработка средств защиты, перекрывающих выявленные каналы НСД и обеспечивающих заданный уровень безопасности (прочности защиты) информации;
- разработка средств функционального контроля системы, повышения достоверности и резервирования информации;
- создание или использование готовых средств централизованного контроля и управления безопасностью информации в данной системе;
- оценка уровня ожидаемой эффективности (прочности) защиты на предмет соответствия заданным требованиям;

б) на этапе эксплуатации:

- контроль и поддержка функционирования системы безопасности информации в данной компьютерной системе;



- своевременное предупреждение, обнаружение и блокировка НСД;
- регистрация и учет всех обращений к защищаемой информации, документирование, ведение статистики и прогнозирование НСД.

Выводы

Анализ результатов исследований и работ по обеспечению безопасности информации в компьютерных системах с позиций данной концепции позволяет предположить следующие причины уязвимости многих проектов:

1) отсутствие четких и ясных определений предмета и объектов защиты, а также потенциальных угроз, непосредственно прилагаемых к предмету и объекту защиты;
2) выбор концепции построения защиты, основанной на классификации средств защиты по технологии реализации (аппаратной, программной, физической и т.д.), а не по выполняемым функциям защиты;
3) отсутствие методов расчета прочности средств защиты;
4) отсутствие анализа защиты на предмет образования замкнутого контура защиты и его прочности;
5) уменьшение роли или полное пренебрежение временными факторами: быстродействием обнаружения и блокировки НСД, временем преодоления преграды нарушителем.

Предложенные принципы, по мнению автора, позволяют четко и ясно поставить задачу и найти пути ее решения: определить предмет и объект защиты, потенциальные угрозы, построить на пути нарушителя строгую систему равнопрочных и взаимосвязанных преград с возможностью получения расчетных характеристик ее ожидаемой эффективности.

В.Мельников

Литература:

1. Герасименко В.А. Проблемы защиты данных в системах их обработки. Зарубежная электроника №12, 1989. — М.: Радио и связь.
2. Першин А. Организация защиты вычислительных систем. КомпьютерПресс №10, 1992.
3. Булаиов В.Д., Козырь В.И., Коиашев В.В. О возможном подходе к комплексиой оценке защищеииости объекта АСУ. Сб. статей ВСРЭ сер. СОИУ вып. 23, 1989.
4. Мельииков Ю.Н. Достовериость иифомации в сложиых системах. — М.: Советское радио, 1973.
5. Шураков В.В. Надежиость программиого обеспечеиия систем обработки даииых. — М.: Статистика, 1987.

Дизассемблирование: как это делается

В этой статье мне хочется рассказать о проблемах, возникающих при дизассемблировании, и о том, как эти проблемы можно решить.

На мой взгляд, большинство программистов делятся на два класса. Первые считают, что не имея исходного текста, с программой разобраться невозможно. Вторые уверены, что текст, выданный дизассемблером, можно тут же ассемблировать и что полученная при этом программа будет работать так же, как исходная. Первые, естественно, даже не пытаются дизассемблировать, так как программируют на Бейсике и побаиваются ассемблера. Вторые пытаются, но их, как правило, ждет разочарование. Текст, выданный дизассемблером, либо вообще не поддается реассемблированию, либо получаемая программа ведет себя совсем не так, как хочется.

И все же те, кто смело берется за дизассемблирование, гораздо ближе к получению какого-то результата. Небольшие программы (примерно 1-2 килобайта) могут быть вполне корректно дизассемблированы с помощью популярного дизассемблера SOURCER. Что же касается больших программ, то истина, на мой взгляд, состоит в том, что *полное, автоматическое дизассемблирование невозможно*; над текстом, который выдает дизассемблер, нужно довольно долго работать, прежде чем его повторное ассемблирование даст работоспособную программу.

В дальнейшем я постараюсь рассказать о тех приемах, которые превращают "плохой" текст в "хороший", то есть в текст, который не только дает корректно работающую программу при ассемблировании, но и позволяет себя изменить с целью усовершенствования исходной программы.

Техника дизассемблирования сильно зависит от того, какой дизассемблер применяется. Чтобы сделать изложение более конкретным, я расскажу о своем опыте дизассемблирования довольно большой .COM-программы (графического редактора IMAGE 72) с помощью дизассемблера DisDoc 2.3. Все примеры, приведенные в тексте, взяты из листинга, выданного дизассемблером, и действительно имели место в жизни.

Почему DisDoc?

SOURCER — это название знают все, кто хотя бы краем уха слышал о дизассемблировании. Считается, что это дизассемблер замечательный, мощный, не имеющий конкурентов. Я думаю, что слухи об огромных преимуществах SOURCER сильно преувеличены. У меня сложилось такое впечатление, что при дизассемблировании небольших программ (до 7 Кбайт) SOURCER предпочтительнее. Когда программа велика (в моем случае — 58 Кбайт), SOURCER работает очень медленно и, на мой взгляд, не дает никаких преимуществ.

Выбор дизассемблера DisDoc 2.3 был для меня во многом случаен. Начиная работу, я получил тексты на ассемблере как с помощью SOURCER (версия 3.07), так и с помощью DisDoc 2.3. Затем оба текста после устранения очевидных ошибок были ассемблированы. И вот, то, что было выдано SOURCER'ом, повисло сразу, а то, что выдал DisDoc 2.3, перед зависанием вывело на экран несколько линий. Это и определило выбор. В процессе работы я не раз имел возможность оценить основное преимущество дизассемблера DisDoc — интуитивно понятный, неизощренный, удобный и компактный листинг.

Чтобы понять дальнейшее, необходимо познакомиться с отрывком из листинга, который выдает DisDoc 2.3.

```
         mov   cx,WORD PTR ds:d02453   ;02430
b02430:  add   cx,bx                   ;02434
         mov   bx,99e7h                ;02436
         mov   dx,WORD PTR ds:d02449   ;02439
         mov   al,BYTE PTR ds:d02446   ;0243d
         call  s383 ;<09060>           ;02440
         push  cs                      ;02443
         pop   ds                      ;02444
         ret                           ;02445
;-----------------------------------------------
d02446   db    00                      ;02446 .
d02447   db    00,00                   ;02447 ..
d02449 . db    00,00                   ;02449 ..
```

В поле комментариев указано смещение, которое имела данная инструкция в исходной программе. Например, если в исходной программе, подвергаемой дизассемблированию, вы посмотрите отладчиком смещение 02434, то там окажется инструкция add cx,bx — на это можно положиться! Очень хороши названия меток и элементов данных. По ним сразу можно понять, какое смещение они имели в исходной программе. Например, метка b02430 имела смещение 02430, элемент данных d02446 имел смещение 02446 и т.д. То же самое относится и к подпрограммам. После вызова подпрограммы в треугольных скобках указано смещение, которое имела эта подпрограмма в исходной программе. Например, подпрограмма s383 начиналась в исходной программе со смещения 09060. Такая организация листинга позволяет сохранить однозначное соответствие с исходной программой, что дает возможность проверить отладчиком сомнительные куски кода и данных, сравнить текст, выданный дизас-

семблером, с тем, что есть на самом деле. Это поистине драгоценная возможность.

Нужно сказать, что DisDoc имеет большие недостатки, о которых речь еще впереди, и, следовательно, применение того или иного дизассемблера — дело вкуса. В любом случае обязательно встретятся

Фундаментальные проблемы

1. Проблема OFFSET

Предположим, что в тексте, который выдал дизассемблер, есть такой фрагмент:

```
        mov   ax,bx                  ;1
        shl   ax,1 ;004bc            ;2
        mov   si,8429h               ;3
        add   si,ax                  ;4
        push  WORD PTR [si]          ;5
```

Что засылается в регистр si в третьей строчке — число 8429h или смещение некой метки? На этот вопрос отвечает пятая строчка, из которой видно, что регистр si используется для косвенной адресации. Значит, исправленный фрагмент должен выглядеть следующим образом:

```
        mov   ax,bx                  ;1
        shl   ax,1 ;004bc            ;2
        mov   si,OFFSET d08429       ;3
        add   si,ax                  ;4
        push  WORD PTR [si]          ;5

        .............................

d08429  db    0ff,0ff,0f6            ;8429
        db    0ff,0d8,0ff,0a6,0ff,60 ;0842c .....'
```

Возможно, здесь у многих возникнет сомнение — нужно ли заменять число на соответствующий OFFSET — ведь в заново ассемблированной программе данные должны иметь то же смещение? К сожалению, это не так. Во-первых, мы обычно не знаем, какой ассемблер применялся при транслировании оригинального текста, а коды, полученные с помощью разных ассемблеров, будут иметь разную длину, что приведет к изменению смещений. Например, команда AND CX,0007h транслируется MASM 5.1 и TASM 1.01 как 83E107 и занимает 3 байта. Но эта же команда может быть транслирована как 81E10700 и занимать 4 байта. Во-вторых, даже если при повторном ассемблировании полученного текста смещения сохранятся, программа не поддастся модификации, так как при вставке какого-либо фрагмента кода тут же изменятся идущие за ним смещения и все "развалится".

Итак, замена непосредственных значений смещений смещениями OFFSET позволяет склеить программу, делает ее пригодной для модификации.

Разобранный пример достаточно примитивен. Попробуем рассмотреть более сложные ситуации и исследуем следующий фрагмент текста, выданный дизассемблером:

```
         mov   bx,9006h                 ;08f66
b08f75:  mov   WORD PTR ds:d087d0,bx    ;08f75
         ..........................
         call  WORD PTR cs:d087d0       ;08fc3
         ..........................
;------------------------------------------
         push  dx                       ;09006
         call  s419 ;<099a3>            ;09007
         mov   al,BYTE PTR [si]         ;0900a
         mov   BYTE PTR [si],0ffh       ;0900c
         pop   dx                       ;0900f
         ret                            ;09010
```

Здесь возникает тот же вопрос — что такое 9006h в первой строчке — смещение или просто число? Ответить помогает информация, размещенная в поле комментариев. Мы уже говорили о том, что числа, помещенные в этом поле, представляют собой смещения инструкций в исходной программе. Нетрудно догадаться, что в приведенном фрагменте осуществляется косвенный вызов подпрограммы, и, следовательно, 9006h — это смещение, а не число. Фрагмент должен быть исправлен так:

```
         mov   bx,OFFSET d09006         ;08f66
         ................................
;------------------------------------------
d09006:  push  dx                       ;09006
         ................................
         ret                            ;09010
```

Рассмотрим еще один пример косвенного вызова подпрограммы, в котором OFFSET попадает в область данных:

```
s390     proc  near
...........................................
         mov   ax,WORD PTR [bx+8792h]  ;092c7
         mov   WORD PTR ds:d087d2,ax   ;092cb
...........................................
         call  WORD PTR cs:d087d2      ;093c8
         ret                           ;093d4
;------------------------------------------
         ror   ah,1 ;093d5             ;LO]-->[HI..LO]-->[HI
         jb    b093da ;093d7           ;Jump if <  (no sign)
         ret                           ;093d9
b093da:  inc   si                      ;093da
         ret                           ;093db
```

Чтобы выяснить, что представляет собой 8792h, нужно посмотреть в область со смещениями, близкими к этому числу. Приведем соответствующий фрагмент, выданный дизассемблером:

```
d08790   db    00,00,0d5,93            ;08790 ......
```

Видно, что смещению 08792 соответствует слово 93d5. Теперь остается заметить, что со смещения



093d5 в исходной программе начинается фрагмент повисшего кода

```
        ror   ah,1 ;093d5 !!!!!!        ;LO]-->[Hi..LO]-->[Hi
        jb    b093da ;093d7             ;Jump if <  (no sign)
        ret                             ;093d9
b093da: inc   si                        ;093da
        ret                             ;093db
```

Следовательно, весь разобранный пример — это хитроумный косвенный вызов подпрограммы. Исправленный фрагмент должен выглядеть так:

```
s390    proc  near
..........................................
        mov   ax,WORD PTR [bx+OFFSET d08792]  ;092c7
        mov   WORD PTR ds:d087d2,ax   ;092cb
..........................................
        call  WORD PTR cs:d087d2      ;093c8
        ret                           ;093d4
;-----------------------------------------
d093d5: ror   ah,1 ;093d5             ;LO]-->[Hi..LO]-->[Hi
        jb    b093da ;093d7           ;Jump if <  (no sign)
        ret                           ;093d9
b093da: inc   si                      ;093da
        ret                           ;093db
..........................................
d08790  db    00,00                   ;08790 ......
d08792  dw    OFFSET d093d5           ;08792
```

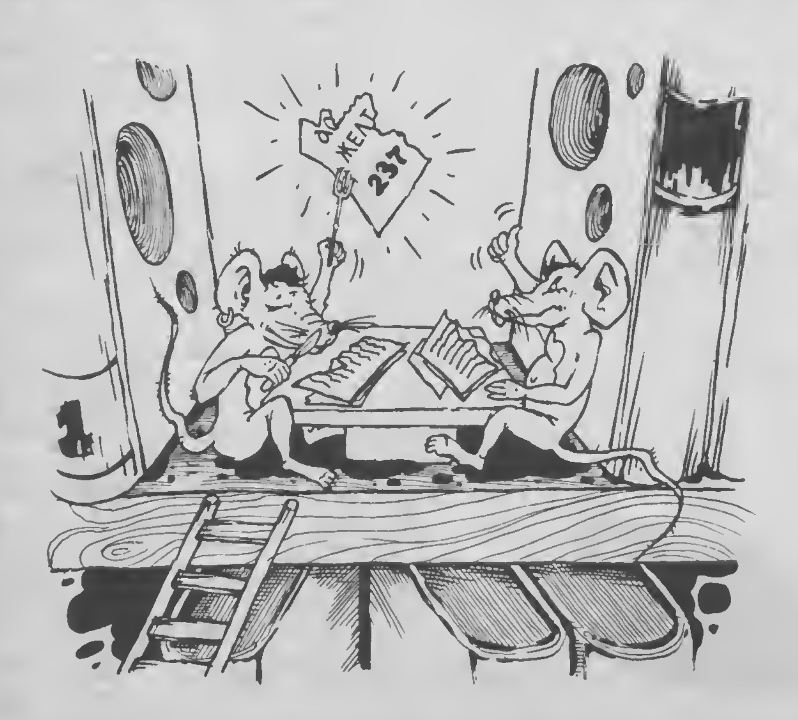
Здесь я предвижу большие возражения. Мне скажут, что все это можно интерпретировать иначе, что мои доказательства неубедительны и т.д. С этим я совершенно согласен. Более того, эти доказательства неубедительны и для меня. Гораздо сильнее убеждает то, что программа после ассемблирования работает! Дизассемблирование, как и отладка программ, — процесс интуитивный. Опытный человек испытывает особое удовольствие от того, что его немотивированные догадки впоследствии подтверждаются. Как часто мысль, пришедшая в автобусе, во сне, в компании, в самой неподходящей обстановке, — оказывается верной!

Завершим этот пункт еще одним достаточно хитрым примером. В тексте, который выдал дизассемблер, встретился такой фрагмент:

```
        mov   bx,4f71h                  ;0522b
b0522e: pop   ax                        ;0522e
        cmp   ax,bx                     ;0522f
        jnz   b0522e ;05231             ;Jump not equal(ZF=0)
        mov   BYTE PTR ds:d041f4,00     ;05233
        push  ax                        ;05238
        ret                             ;05239
        ..............................
        call  s229 ;<04fc4>             ;04f71
```

Возникает все тот же вопрос — что такое 4f71h — число или смещение? Чтобы на него ответить, нужно понять, что делает этот участок программы. Давайте попробуем в этом разобраться. Очевидно, из стека выталкивается число, сравнивается с 4f71h и, если нет равенства, выталкивается следующее число. Если число равно 4f71h, то оно снова заталкивается в стек и происходит возврат из подпрограммы. Но куда? Ясно, что в то место, смещение которого в исходной программе было равно 4f71h. Как видно из текста, в этом месте стоял вызов подпрограммы s229. Значит, таким необычным образом вызывается подпрограмма, и 4f71h — это смещение! Исправленный фрагмент должен выглядеть так:

```
        mov   bx, OFFSET b04f71           ;0522b
b0522e: pop   ax                          ;0522e
        cmp   ax,bx                       ;0522f
        jnz   b0522e ;05231        ;Jump not equal(ZF=0)
        mov   BYTE PTR ds:d041f4,00 ;05233
        push  ax                         ;05238
        ret                             ;05239
        ..............................
b04f71: call  s229 ;<04fc4>              ;04f71
```



2. Как отличить данные от команд

Любой дизассемблер путает данные и команды. Особенно это относится к .COM-программам, где все перемешано. Рассмотрим простой пример:

```
        pop   cx                        ;03e56
        ret                             ;03e57
;-----------------------------------------
        add   BYTE PTR [bx+si],al       ;03e58
        add   BYTE PTR [bx+si],al       ;03e5a
m03e5c: mov   BYTE PTR ds:d05830,01     ;03e5c
```

В этом фрагменте встретились две вычурных, "повисших" инструкции:

```
        add   BYTE PTR [bx+si],al       ;03e58
        add   BYTE PTR [bx+si],al       ;03e5a
```

Сверху они ограничены инструкцией возврата из подпрограммы ret, а снизу — меткой m03e5c. Ясно, что эти инструкции могут быть только данными. После переделки приведенный фрагмент должен выглядеть так:

```
        pop   cx                        ;03e56
        ret                             ;03e57
;-----------------------------------------
d03e58  dw    0                         ;03e58
d03e5a  db    0                         ;03e5a
d03e5b  db    0
m03e5c: mov   BYTE PTR ds:d05830,01     ;03e5c
```

Тут возникает еще один вопрос: почему в одном случае стоит dw, а в другом — db? Ответ содержится в тексте, который выдал дизассемблер. Там можно найти такие инструкции:

```
        mov   si,WORD PTR ds:d03e58     ;03dd0
        mov   bl,BYTE PTR ds:d03e5a     ;03dd4,
```

откуда следует, что d03e58 рассматривается как слово, а d03e5a — как байт.

Рассмотрим чуть более сложный, но тем не менее очень характерный пример:

```
b03f53: cmp   al,05                    ;03f53
        jnz   b03f6b ;03f55            ;Jump not equal(ZF=0)
        ..............................
        ret                            ;03f69
;----------------------------------------
        add   BYTE PTR [si],bh         ;03f6a
        push  es                       ;03f6c
        jnz   b03f79 ;03f6d            ;Jump not equal(ZF=0)
```

В приведенном фрагменте текста метка b03f6b отсутствует. Между тем эта метка должна “разрубить“ пополам инструкцию add BYTE PTR [si],bh, которая начинается в оригинальной программе, подвергаемой дизассемблированию, со смещения 03f6a. Выход здесь может быть только один — смещению 03f6a должен соответствовать байт данных, а искомая инструкция будет действительно начинаться со смещения 03f6b. Исправленный фрагмент выглядит так:

```
b03f53: cmp   al,05                    ;03f53
        jnz   b03f6b ;03f55            ;Jump not equal(ZF=0)
........................................
        ret                            ;03f69
;----------------------------------------
d03f6a  db    0                        ;03f6a
b03f6b: cmp   al,06h                   ;03f6b
        jnz   b03f79 ;03f6d            ;Jump not equal(ZF=0)
```

Нужно сказать, что путаница с данными и инструкциями возникает довольно часто. SOURCER способен выдавать целые метры бессмысленных инструкций. В этом отношении DisDoc 2.3 ведет себя лучше.

3. Зависимость от транслятора

Программисты на ассемблере склонны пренебрегать правилами хорошего тона, нарушать все мыслимые табу, и это создает дополнительные трудности при дизассемблировании. В качестве примера приведем фрагмент кода, выданного дизассемблером:

```
s25     proc  near
        inc   cx                       ;0086b
        add   di,bp                    ;0086c
        adc   si,00                    ;0086e
        add   dx,si                    ;00871
        push  di                       ;00873
        shl   di,1 ;00874              ;Multiply by 2's
        adc   dx,00                    ;00876
        pop   di                       ;00879
        ret                            ;0087a
```

Этот фрагмент представляется совершенно невинным, и действительно, он дизассемблирован правильно. Вся беда в том, что программист задумал использовать самогенерируемые коды — автоматически изменять команды данного фрагмента в процессе выполнения программы, то есть резать по живому. Оказывается, в программе есть еще такой кусок:

```
        mov   di,086bh                 ;007f8
        ..............................
        mov   BYTE PTR [di],4ah        ;00800
        mov   BYTE PTR [di+07],0f1h    ;00803
        mov   BYTE PTR [di+0ch],0d1h   ;00807
        ..............................
        ret                            ;00815
```

Рис.1



Так как di используется для косвенной адресации, нам прежде всего необходимо заменить 086bh на соответствующий OFFSET b0086b и пометить этой меткой начало подпрограммы s25:

```
s25     proc  near
b0086b: inc   cx                       ;0086b
        ..............................
```

Далее следует понять, что делают инструкции, приведенные на рис. 1, с подпрограммой s25. Пусть эта подпрограмма ассемблирована с помощью TASM 1.01. Выданный ассемблером код будет таким, как показано на рис. 2.

```
41       INC   CX               41       INC   CX
03F0     ADD   DI,BP            01EF     ADD   DI,BP
830600   ADC   SI,0000          830600   ADC   SI,0000
03D6     ADD   DX,SI            01F2     ADD   DX,SI
57       PUSH  DI               57       PUSH  DI
01E7     SHL   DI,1             01E7     SHL   DI,1
830200   ADC   DX,0000          8302000  ADC   DX,0000
5F       POP   DI               5F       POP   DI
C3       RET                    C3       RET
```

Рис.2 Рис.3

Но вся беда в том, что исходная программа была ассемблирована другим ассемблером и имеет вид, показанный на рис. 3. Как видно из сравнения рис. 2 и 3, TASM 1.01 и неизвестный ассемблер транслируют инструкции ADD по-разному, и это приводит к катастрофическим последствиям. Действительно, посмотрим, как воздействует участок кода, показанный на рис. 1 (в котором 086bh заменено на OFFSET b0086b), на подпрограмму s25, транслируемую TASM (рис. 4) и неизвестным ассемблером (рис. 5).

```
4A       DEC   DX               4A       DEC   DX
03F0     ADD   DI,BP            01EF     ADD   DI,BP
830600   ADC   SI,0000          830600   ADC   SI,0000
03F1     ADD   SI,CX  ;!!!      01F1     ADD   CX,SI  ;!!!
57       PUSH  DI               57       PUSH  DI
01E7     SHL   DI,1             01E7     SHL   DI,1
83D100   ADC   CX,0000          83D100   ADC   CX,0000
5F       POP   DI               5F       POP   DI
C3       RET                    C3       RET
```

Рис.4 Рис.5

Сравнение рис. 4 и 5 показывает, что логика работы программы меняется в зависимости от того, какой ассемблер применялся.

Как выкрутиться из этой ситуации, если нужного ассемблера нет под рукой? Самый простой, но не очень красивый путь — поставить “заплатку“. Чтобы можно было использовать TASM, подпрограмма s25 должна выглядеть так:

```
s25     proc  near
d0086b: inc   cx                       ;0086b
        add   di,bp                    ;0086c
```

```
        adc   sl,00                    ;0086e
        db    01,0f2                   ;add  dx,sl !!!! ;00871
        push  dl                       ;00873
        shl   dl,1 ;00874              ;Multlply by 2's
        adc   dx,00                    ;00876
        pop   dl                       ;00879
        ret                            ;0087a
```

Особенности и ошибки дизассемблера DisDoc 2.3

К сожалению, DisDoc 2.3 совершает ошибки, иногда регулярные, а иногда редкие, коварные и даже подлые. Самая противная ошибка — случайный пропуск данных — встречается довольно редко. Начнем с того, что встречается очень часто.

1. EQU — кто тебя выдумал?

В коде, выданном дизассемблером, часто попадаются такие загадочные куски:

```
;<00465>
s12      proc  near
d0046c   equ   00046ch
         cmp   bx,5ah                  ;00465
```

Каков смысл присвоения d0046c equ 00046ch? Чтобы выяснить это, нужно отыскать d0046c в тексте. В нашем случае элемент данных d0046c встречается очень далеко от своего первого упоминания — в подпрограмме s321:

```
         mov   ax,0040h                ;06257
;<es = 0040>
         mov   es,ax                   ;0625a
         mov   al,BYTE PTR es:d0046c   ;0625c
         sti    ;06260                 ;Turn ON Interrupts
b06261:  cmp   al,BYTE PTR es:d0046c   ;06261
         jz    b06261 ;06266           ;Jump if equal (ZF=1)
         mov   al,BYTE PTR es:d0046c   ;06268
         dec   cx                      ;0626c
         jnz   b06261 ;0626d           ;Jump not equal(ZF=0)
         pop   ax                      ;0626f
         out   61h,al ;06270       ;060-067:8024 keybrd contrlr
;<es = 0000>
         pop   es                      ;06272
         ret                           ;06273
s321     endp
```

Рис.6

При виде этого текста возникает догадка, что здесь идет взаимодействие с областью данных BIOS. Действительно, в сегментный регистр es засылается число 40, то есть es будет указывать на начало области системных данных с физическим адресом 400. Следующий вопрос — каков смысл адреса 046ch? Легко выяснить, что по этому (физическому) адресу находится счетчик прерываний от таймера. Теперь ясен смысл работы фрагмента на рис. 6 — он дает задержку на число прерываний от таймера, заданное в регистре cx. Но если это так, то d0046c должно быть равно не 46ch, а просто 6ch (либо es должен указывать на нулевой сегмент)! И действительно, если посмотреть это место отладчиком, то станет ясно, что вместо mov al,BYTE PTR es:d0046c в этом тексте обязательно должно стоять mov al,es:006ch.

Итак, чтобы исправить эту ошибку, необходимо:

1. Удалить из начала этой подпрограммы s12 присвоение d0046c equ 00046ch.

2. Переписать приведенный на рис. 6 фрагмент s321 следующим образом:

```
         mov   ax,0040h                ;06257
;<es = 0040>
         mov   es,ax                   ;0625a
         mov   al,BYTE PTR es:006ch    ;0625c
         sti    ;06260                 ;Turn ON Interrupts
b06261:  cmp   al,BYTE PTR es:006ch    ;06261
         jz    b06261 ;06266           ;Jump if equal (ZF=1)
         mov   al,BYTE PTR es:006ch    ;06268
         dec   cx                      ;0626c
         jnz   b06261 ;0626d           ;Jump not equal(ZF=0)
         pop   ax                      ;0626f
         out   61h,al ;06270       ;060-067:8024 keybrd contrlr
;<es = 0000>
         pop   es                      ;06272
         ret                           ;06273
s321     endp
```

Рассмотрим второй пример. В коде, выданном дизассемблером, встретился такой кусок:

```
;<0074e>
s22      proc  near
d0076a   equ   00076ah
d00771   equ   000771h
         call  s24 ;<00791>            ;0074e
          ......   ...   .....   .....
b0076a:  push  cx                      ;0076a
         call  s25 ;<0086b>            ;0076b
         call  s23 ;<00776>            ;0076e
         pop   cx                      ;00771
         dec   bx                      ;00772
```

Поиск элемента данных d0076a окончился неудачей. А d00771 встретился в таком фрагменте:

```
         mov   BYTE PTR ds:b0076a,51h  ;0080b
         mov   BYTE PTR ds:d00771,59h  ;00810
```

Здесь явно идет модификация кода подпрограммы s22. Значит, необходимо заменить d00771 на b00771, пометить этой меткой соответствующую инструкцию в s22 и удалить ненужные начальные присвоения.

Исправленный фрагмент s22 будет выглядеть так:

```
;<0074e>
s22      proc  near
         call  s24 ;<00791>            ;0074e
         ...............................
b0076a:  push  cx                      ;0076a
         call  s25 ;<0086b>            ;0076b
         call  s23 ;<00776>            ;0076e
b00771: .pop   cx                      ;00771
         dec   bx                      ;00772
         ...............................
         mov   BYTE PTR ds:b0076a,51h  ;0080b
         mov   BYTE PTR ds:b00771,59h  ;00810
```

Рассмотрим еще один пример. В начале s32 встретились уже знакомые псевдооператоры:

```
;<00bf7>
s32      proc  near
d00c1c   equ   000c1ch
d00c1e   equ   000c1eh
```



Если посмотреть в область со смещениями, близкими к 0c1ch, то там окажется кусок повисшего кода, который может быть только данными:

```
or    al,BYTE PTR [bp+di]     ;00c14
add   WORD PTR [bx+di],ax     ;00c16
add   BYTE PTR [bx+si],al     ;00c18
add   BYTE PTR [bx+si],al     ;00c1a
mov   di,1306h                ;00c1c
add   ax,06c0h                ;00c1f
```

Рис.7

Теперь нужно поискать идентификаторы d00c1c и d00c1e в тексте, выданном дизассемблером. Очень быстро можно найти фрагменты типа

```
mov WORD PTR ds:d00c1c,ax ,    mov WORD PTR ds:d00c1e,ax
```

Значит, ошибка дизассемблера состоит в том, что он перепутал данные и команды и на этой почве сделал два неправильных присвоения equ, попавших в начало подпрограммы s32.

Исправления будут заключаться в следующем:

1. Убрать из начала подпрограммы s32 два псевдооператора equ.

2. Переписать коды на рис. 7 следующим образом:

```
d00c14   db   0a,03,01,01,00,00,:.,00        ;00c14
d00c1c   db   0bf,06                         ;00c1c
d00c1e   db   13,05,0c0,06                   ;00c1e
```

Рассмотрим еще один совсем простенький фрагмент кода:

```
;<01252>
s39      proc  near
d0125d   equ   00125dh
d0125f   equ   00125fh
         dec   bh                       ;01252
         jz    b0124f ;01254            ;Jump if equal (ZF=1)
         xor   ah,ah                    ;01256
         shl   al,1 ;01258              ;Multiply by 2's
         rcl   ah,1 ;0125a              ;CF<--[HI .. LO]<--CF
         ret                            ;0125c
;----------------------------------------
         add   BYTE PTR [bx+si],al      ;0125d
         add   BYTE PTR [bx+si],al      ;0125f
s39      endp
```

Укажем без комментариев, что подпрограмма s39 должна выглядеть так:

```
;<01252>
s39      proc  near
         dec   bh                       ;01252
         jz    b0124f ;01254            ;Jump if equal (ZF=1)
         xor   ah,ah                    ;01256
         shl   al,1 ;01258              ;Multiply by 2's
         rcl   ah,1 ;0125a              ;CF<--[HI .. LO]<--CF
         ret                            ;0125c
;----------------------------------------
d0125d   db    00,00                    ;0125d
d0125f   db    00,00                    ;0125f
s39      endp
```

Теперь подведем итоги. Значки equ называют псевдооператорами. Если говорить о дизассемблере DisDoc 2.3, то это название удивительно точное. Если в тексте встретится equ — то ошибка рядом. Между тем, иногда DisDoc 2.3 употребляет equ вполне корректно. Так что будьте бдительны и не дайте себя обмануть.

2. Случайные ошибки

Иногда поведение дизассемблера трудно объяснить. Например, он выдает

```
add   WORD PTR ds:d96be3,07   ;038b6
shr   WORD PTR ds:d96be3,cl   ;038bb ;Divide by 2's
```

вместо

```
add   WORD PTR ds:d06bf3,07   ;038b6
shr   WORD PTR ds:d06bf3,cl   ;038bb ;Divide by 2's ,
```

теряет или искажает куски данных. К счастью, это происходит достаточно редко.

А.Крупник



Импорт объектов из внешней программы на C++

Уважаемая редакция!

Прочитав в КомпьютерПресс № 1'92 статью С.Кучерова "Импорт объектов из внешней программы на Turbo Pascal", я попытался реализовать то же самое на C++. Хочу поделиться тем, что у меня получилось.

Должен признаться, описанная в статье реализация этой идеи понравилась мне не во всем. И основной недостаток, на мой взгляд, в том, что программе пользователя приходится в начале работы предпринимать некоторые обязательные действия; хотелось бы, чтобы пользователю не надо было ни о чем заботиться (тогда он, кстати, был бы лишен возможности сделать это неправильно).

Задачу для себя я сформулировал так.

Есть главная программа, которая может запускать приложения, написанные пользователем, и предоставлять им доступ к своим объектам.

Пользователю предоставляются:

1) загрузочный модуль главной программы;
2) файлы заголовков с описаниями доступных приложению классов;
3) объектный модуль, обеспечивающий доступ к объектам главной программы.

От пользователя, создающего собственное приложение, требуется лишь включить указанный объектный модуль при компоновке задачи. Никаких других действий от пользователя не требуется; объекты из главной программы доступны программе пользователя так же, как и собственные объекты пользовательской программы.

Предположим для примера, что в главной программе у нас есть класс Base, который мы хотим импортировать в приложение:

```
class Base {
  public:
    virtual const char* nameOf();
};
const char* Base::nameOf()  { return "Class Base"; }
```

Вместо того чтобы копировать VMT в приложение (как делает С.Кучеров), мы будем создавать объекты, у которых указатель на VMT содержит адрес VMT главной программы (в Borland C++ для этого надо включить опцию компилятора Far Virtual Tables). Возникает вопрос: как заставить конструктор Base поместить в объект из Child адрес VMT из Main? Для этого проще всего использовать конструктор из Main для создания объекта в Child. Поскольку будет вызываться конструктор, принадлежащий Main, он поместит в создаваемый объект указатель на VMT из Main, что и требовалось. Беда в том, что адрес конструктора нельзя получить средствами C++, поэтому для передачи его в Child придется пойти на маленькую хитрость.

Добавим в класс Base еще один компонент

```
void far* Base::operator new (unsigned size, void far* p)
{ return p; }
```

и опишем функцию

```
void huge MakeBase(void far* Where)
{
  new (Where) Base;
}
```

Определенная таким образом функция MakeBase будет принимать указатель и создавать по этому адресу объект типа Base. Действительно, при ее выполнении сначала будет выполнен оператор new, переопределенный нами выше. Он практически ничего не делает, только возвращает указатель, переданный ему в качестве параметра (параметр size — обязательный для любого оператора new; посредством size оператору new автоматически передается размер создаваемого объекта, поэтому при вызове new не нужно явно задавать size). Затем полученный от new указатель передается конструктору Base в качестве this, и конструктор создает по этому адресу объект типа Base.

MakeBase объявлена huge, так как, во-первых, она будет вызываться посредством дальнего вызова из

Child, во-вторых, ей при вызове необходимо установить регистр ds на сегмент данных программы Main. Следует отметить, что всякая функция из Main, которая будет использоваться из Child, должна быть объявлена huge (или _loadds far), иначе она не сможет пользоваться статическими данными.

Адрес функции MakeBase мы можем средствами C++ передать в Child. Нам придется написать для Child фиктивный конструктор, который будет просто вызывать функцию MakeBase, передавая ей параметр this.

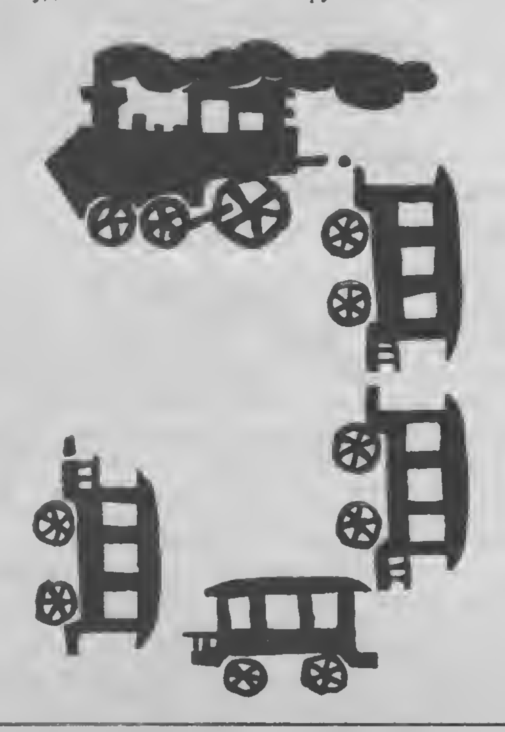
Заметим, что при таком переопределении оператора new нам придется добавить в Base описание обычного оператора new:

```
inline void* Base::operator new (unsigned size)
       { return ::new Base; }
```

Его действие заключается только в вызове стандартного new. Необходимость в его явном описании объясняется тем, что конструктор Base() всегда неявно вызывает оператор new (unsigned int); поскольку мы переопределили старый оператор, при описании конструктора у нас возникала бы ошибка. Определение нового new, который идентичен старому, спасает положение.

Адрес функции MakeBase будет первым элементом таблицы настройки, адрес которой мы передадим в Child при его запуске. Помимо этого, мы должны позаботиться о функциях управления кучей.

Мне показалось (хотя это, конечно, вопрос спорный), что вместо переустановки значений управляющих кучей переменных в Main удобнее использовать функции управления памятью из программы Child. Другими словами, при запуске Child мы передаем в Main адреса функций farmalloc, farfree и т.д. из Child, и Main будет использовать эти функции вместо своих собственных, пока выполняется Child. Разумеется, этот способ более громоздкий, чем предложенный в статье, но, по-моему, он безопаснее. В класс Base я включил функцию testHeap, которая совершает некоторые операции с кучей, а затем проверяет ее.



Еще одна важная деталь, необходимая таблице настройки, — какой-либо отличительный признак, позволяющий Child удостовериться, что ей передана именно таблица настройки. Таким признаком может служить, например, строка, содержащая определенный текст.

Кажется, мы перечислили все, что должна содержать таблица настройки. Она будет выглядеть приблизительно так:

```
class TuneTable {
  public:
    // отличительный признак
    char far* sign;
    // указатель на функцию инициализации Base
    void huge (* MakeBasePtr)(void far*);
    // указатели на функции управления кучей
    void far* huge (* farmallocPtr)(unsigned);
    void far* huge (* farfreePtr)(unsigned);
    void      huge (* farheapcheckPtr)(void);
    // . . . . .
  TuneTable();       // конструктор
};
```

Теперь нам необходимо передать адрес таблицы из Main в Child.

Как известно, при запуске процесса-потомка родитель имеет право передать ему несколько аргументов, которые суть ASCIIZ-строки. Преобразуем дальний адрес таблицы настройки в строку из восьми символов и передадим ее в Child вторым аргументом вызова (первым аргументом всегда передается полное имя вызванной программы). Эти действия выполняются в вызывающей программе при помощи функции ultoa, которая принимает 3 аргумента: длинное беззнаковое целое — число, которое следует преобразовать в строку; строку, куда нужно поместить результат; и целое — основание системы счисления.

Методы класса Base, функцию MakeBase и настроечную таблицу я разместил в модуле EXPORT, который должен компоноваться с программой Main.

С программой Child будет компоноваться модуль IMPORT, который позаботится о корректной передаче процессу Child всего необходимого для использования класса Base. Модуль IMPORT включает те же файлы заголовков, что и EXPORT: base.h и tunetabl.h, которые содержат описания классов Base и TuneTable соответственно. Таким образом, структура этих классов для Main и Child идентична, чего нельзя сказать о методах.

Начнем с того, что TuneTable::operator new в модуле IMPORT получает адрес таблицы настройки при помощи второго аргумента вызова (_argv[1]). Удостоверившись, что адрес не нулевой, он передаст его конструктору, который, в свою очередь, проверит при помощи строки-признака, действительно ли ему передана настроечная таблица. Затем конструктор устанавливает адреса функций управления памятью, которые будут использоваться из Main. Наконец, он скрывает следы нашей деятельности по передаче адреса таблицы: переносит адрес третьего аргумента на место второго и т.д.,

а затем уменьшает на единицу количество аргументов. После этого программа пользователя получит нормальный список аргументов, в котором никакого адреса настроечной таблицы нет.

Затем мы создаем статический объект theTablePtr типа 'указатель на TuneTable' и инициализируем его при помощи new, обеспечивая вызов оператора new и конструктора перед вызовом процедуры main из процесса Child.

Конструктор для Base использует находящийся в таблице адрес функции MakeBase для того, чтобы инициализировать создаваемый объект. Теперь любой объект типа Base в Child будет создаваться при помощи функции MakeBase из Main, то есть в него будет помещен нужный адрес VMT.

Методы класса Base определены, на первый взгляд, довольно странно, например

```
const char far* huge Base::nameOf() { return this->nameOf(); }
```

Дело в том, что, хотя мы и подставили в Base адрес нужной нам VMT, это решает не все проблемы. Адрес VMT используется только тогда, когда функция вызывается посредством указателя, к примеру

```
Base* bPtr = new Base;
bPtr->nameOf();
```

Если же функция вызывается непосредственно:

```
Base b;
b.nameOf();
```

то компилятор подставит прямое обращение по адресу функции. Наше описание заставит его все-таки использовать VMT. Запустив эту программу, можно убедиться, что здесь не происходит бесконечной рекурсии, как могло бы показаться.

Кроме того, в модуле IMPORT содержатся функции управления памятью, которые будут использоваться из Main. Эти функции просто вызывают стандартные farmalloc, farfree и т.д.; единственное, зачем нужно такое переопределение, — это определить их как huge. Надо только не забыть после завершения процесса Child восстановить содержимое настроечной таблицы. Это делается оператором

```
theTable = TuneTable();
```

в программе Main. Возможно, было бы удобно выделить в отдельную функцию все действия, связанные с подготовкой и запуском процесса Child, но я уж не стал этого делать.

При компиляции надо иметь в виду следующее. Во-первых, как уже говорилось, Main и Child должны компилироваться с опцией Far Virtual Tables (Options|Compiler|C++ Options). Для компиляции Main надо использовать модель памяти с дальними вызовами функций. Child может создаваться с использованием любой модели. Кроме того, при компиляции Main следует отключить опцию Test Stack Overflow (Options|Compiler|Entry/Exit Code). Это связано с тем, что при выполнении Child функции из Main будут использовать стек Child. В разных моделях памяти проверка переполнения стека выполняется по-разному, поэтому, если модели Main и Child не совпадают, то корректный стек Child может показаться функции из Main безнадежно испорченным. Положение не спасет даже установка в Main значений _stklen и __brklvl из Child. Так что, если действительно важно контролировать стек, придется написать собственную функцию, которая проверяла бы модель памяти Child и в зависимости от этого выбирала критерий проверки.

Для того чтобы создать собственное приложение, пользователю надо иметь файл заголовка base.h и набор объектных модулей IMPORT для каждой модели памяти. Ни о чем специально заботиться не надо, нужно только не забыть установить Far Virtual Tables. Собственно этого я и хотел добиться.

Если у класса Base будет не один, а несколько конструкторов (или потребуется передать в приложение несколько классов), то для каждого конструктора нужно будет создать функцию, подобную MakeBase, и включить ее адрес в настроечную таблицу. Надо помнить, что ВСЕ методы передаваемых классов следует объявлять virtual.

Исходные тексты программ я компилировал при помощи Borland C++ 2.0 и убедился, что они работают.

К.Шмидт

От редакции:
Исходные тексты программ к данной статье вы можете найти в сети Relcom по адресу: novex@bcn.msk.su, password = soft

Простой многопроцессный монитор для программ на Turbo Pascal

При создании приложений нередко возникает задача управления или обработки информации одновременно и параллельно по нескольким относительно независимым алгоритмам. Особенно часто эта проблема встает при разработке программ управления внешними объектами или группами объектов, когда алгоритмы управления отдельными объектами независимы, но все объекты должны обслуживаться одновременно. В этом случае исполнение отдельных алгоритмов может синхронизироваться как с внешними (сигналы от объекта управления), так и с внутренними событиями (программная задержка) для программы управления, приводя к появлению состояний ожидания. Другой пример — интерактивные программы, одновременно работающие с несколькими каналами ввода/вывода (клавиатура, дисплей, мышка, диск, принтер и т.д.).

Подобные задачи естественно приводят к необходимости организации нескольких параллельно функционирующих вычислительных процессов внутри управляющей программы. Некоторые современные операционные системы позволяют организовать многозадачность в режиме разделения времени. Однако их использование далеко не тривиально, связано с определенными требованиями к вычислительной системе и потому не всегда оправданно, особенно для реализации простых программ.

В данной статье предлагается способ организации многопроцессного функционирования для программ, разработанных в системе программирования Turbo Pascal версии 6.0. При этом используются только собственные средства и возможности системы Turbo Pascal.

"Лобовой" подход к созданию Pascal-программ, реализующих на-



бор параллельно работающих ветвей алгоритма управления (примененный авторами в одной из разработок), заключается в следующем. Каждая ветвь алгоритма реализуется отдельной процедурой. Главная ("мониторная") программа последовательно вызывает каждую процедуру. В свою очередь процедура при вызове должна совершить необходимые вычисления или действия в соответствии с шагом алгоритма, а при возникновении ситуации ожидания наступления какого-либо события запомнить свое текущее состояние и вернуть управление главной программе. После аналогичных вызовов других процедур рассматриваемая процедура вызывается снова, в соответствии с текущим состоянием алгоритма (запомненным в предыдущем вызове) управление передается на соответствующий участок процедуры и производится проверка наступления ожидаемого события. Если событие не произошло, следует выход из процедуры. Если же ожидаемое событие произошло, осуществляется переход к следующей фазе алгоритма, выполняются необходимые действия вплоть до следующей точки ожидания, где процесс повторяется.

Как показывает опыт, такой подход приводит к огромному усложнению программы уже для алгоритмов средней сложности. Программа обрастает огромным числом операторов case ProcessState of... и GoTo. Даже простой алгоритм становится в таком виде совершенно неузнаваемым, программа полностью теряет читабельность и ясность, несмотря на подробные комментарии. Программист практически лишается возможности использовать изящные средства языка Pascal для структурирования программы.

Идея другого подхода, предлагаемого нами, состоит в том, чтобы реализовать каким-либо оператором состояние ожидания наступления события одним из процессов так, чтобы не блокировать выполнение других процессов.

В обычных "последовательных" Pascal-программах оператор ожидания реализуется следующей очевидной конструкцией:

```
repeat until Event;
```

где Event — логическое выражение или функция типа boolean. Выполнение такого оператора в одной из процедур "параллельной" программы приведет к полному блокированию работы других ветвей алгоритма.

Применительно к параллельным программам предлагается модифицировать вышеприведенный оператор следующим образом:

```
repeat Branch until Event;
```

где Branch — специальная процедура программы монитора, которая передает управление следующему по порядку процессу. Эта процедура с точки зрения вызова и возврата является обычной Pascal-процедурой. При этом передача управления очередному процессу осуществляется просто возвратом из процедуры Branch в программу соответствующего процесса. Таким образом, после вызова всех процессов управление снова вернется в данный процесс, а для него это будет выглядеть просто как возврат из вызванной внешней процедуры Branch. Для каждого из процессов процедура Branch является "пустышкой", не производящей никаких действий по обработке информации.

Физически такое взаимодействие реализуется следующим образом. Процесс — это процедура языка Pascal без параметров. Каждому процессу выделена своя не-



пересекающаяся с другими область стека. Процедура Branch после вызова сохраняет в специальной таблице текущее значение регистра BP, выбирает из таблицы значение BP для следующего процесса, загружает его в регистр и делает стандартный для Turbo Pascal возврат из процедуры. То, что процедура оперирует с регистром BP, а не SP, объясняется тем, что в процессе возврата из процедуры содержимое BP используется для восстановления значения SP для вызвавшей программы.

Многопроцессный монитор реализован в виде модуля MultiPro, состоящего из трех процедур. Ниже приводится распечатка интерфейсной части модуля.

```
Unit MultiPro;
interface
type
  TProcess = procedure;
const
  MaxNumberOfProcesses              = 16;
  NumberOfActiveProcesses : byte =  0;
var
  FreeStackSize             : word ;
procedure RegistrateProcess
          ( Process : TProcess; StackSize : word);
          {зарегистрировать процесс}
procedure StartProcessing;
   {Запустить все зарегистрированные процессы}
procedure Branch;  {Передать управление следующему процессу}
```

Модуль содержит три процедуры.

Процедура RegistrateProcess служит для регистрации процесса во внутренней таблице монитора. В качестве параметров ей необходимо передать имя процедуры, реализующей соответствующий процесс, и размер пространства стека в байтах, который следует зарезервировать для данного процесса.

Каждый процесс должен быть реализован обычной Pascal-процедурой без параметров. К такой процедуре предъявляются два основных требования:

1. Все состояния ожидания внешних событий должны быть исполнены вышеописанным способом с использованием процедуры Branch.

2. Процедура не должна ни при каких условиях осуществлять "возврат в вызвавшую программу". То есть монитор не позволяет закончить ни один из процессов, но можно завершить выполнение всей программы оператором Halt в любом из процессов.

В начале выполнения программы нужно вызвать процедуру RegistrateProcess необходимое число раз для регистрации всех процессов. После этого для запуска монитора и начала выполнения процессов нужно вызвать процедуру StartProcessing.

Синхронизация и передача данных между процессами осуществляется через глобальные переменные, как для обычных процедур Pascal. При этом не возникает проблем с арбитражем доступа к общим областям, так как моменты передачи управления между процессами детерминированы и определяются самими процессами. Кроме того, возможность для любого из процессов захватить управление на требуемое время (например, в случае появления срочной вычислительной работы) может оказаться полезной в некоторых приложениях.

Длина исходного текста модуля — 80 строк; оттранслированный код занимает в памяти 755 байт, а переменные в сегменте данных — 110 байт.

Ниже приводится распечатка примера использования монитора.

```
program DemoMPro;
{$F+}
uses MultiPro, Crt;
const
  Kbyte = 1024;
  Counter : longint = 0;  {Програминый счетчик}

procedure ReadingKeyboardProcess;
     {Процесс ожидает нажатия клавиши, }
     {читает символ с клавиатуры и     }
     {выдает его на дисплей            }
  var
    Ch : char;
  const
    Esc = chr(27);
  begin
  repeat
    {Ожидание нажатия}
    repeat Branch until KeyPressed;
    Ch := UpCase(ReadKey);
    if Ch = Esc then  {Выход}
      begin
      NoSound;
      Halt(0);
      end
    else {Вывод символа на дисплей}
      begin
      GotoXY(1,2);
      writeln(' Key : ', Ch );
      end;
    until false;
  end; {ReadingKeyboardProcess}

procedure CountingProcess;
     {Процесс увеличивает на единицу }
     {програмный счетчик при каждом }
     {вызове                         }
  begin
  repeat
    Counter := Counter + 1;
    Branch;
    until false;
  end; {CountingProcess}

procedure WritingProcess;
     {Процесс выдает на дисплей     }
     {текущее значение программного }
     {счетчика                      }
  begin
  repeat
    GoToXY(1,3);
    writeln(' Counter = ', Counter);
    Branch;
    until false;
  end;

procedure SoundingProcess;
        {Процесс выдает попеременно   }
        {два тона на динамик          }
var
    i    : longint;
    Flag : boolean;
  begin
  Flag := false;
  repeat
    {Програминая задержка}
    for i := 1 to 100  do Branch;
    Flag := not Flag;
    if Flag then Sound (400) else Sound (800);
    until false;
  end; {SoundingProcess}

begin {DemoMPro}
ClrScr;
writeln(' Press ESC to terminate program. ');
RegistrateProcess(ReadingKeyboardProcess, 2*Kbyte);
RegistrateProcess(CountingProcess,        2*Kbyte);
RegistrateProcess(WritingProcess,         2*Kbyte);
RegistrateProcess(SoundingProcess,        2*Kbyte);
StartProcessing;
end.
```

Программа реализует четыре процесса. Первый из них ожидает ввода символа с клавиатуры и выдает его на дисплей. Второй увеличивает на единицу значение глобальной переменной Counter. Третий выводит текущее значение этой переменной на дисплей. Четвертый выдает на динамик попеременно два тона, причем длительность тона задается при помощи программной задержки, также реализованной с помощью процедуры Branch. Завершается программа нажатием клавиши Escape.

Монитор MultiPro не претендует на законченность и полную универсальность. Это скорее краткая иллюстрация предлагаемого подхода. Для конкретных приложений могут быть разработаны более мощные системы, позволяющие, например, использовать для областей стека процессов не только стек программы, но и большую память; регистрировать, запускать, останавливать и удалять процессы по командам других процессов и т.д. Надеемся, что эта иллюстрация поможет пользователям Turbo Pascal в разработке мощных, универсальных и красивых программ, достойных великолепной системы, с помощью которой они создаются.

Р.Иванцов, А.Пилипенко, А.Шемсудов

Александр Синев:

Пошла муха на базар...

Маленькое исследование на предмет того, что продается и что покупается на российском рынке программного обеспечения.

Несколько вступительных слов. Как и следовало ожидать, софтверный рынок бурно развивается, и сегодня у нас можно *купить* многое из того, что раньше приходилось с трудом *доставать*. Причем сделать это теперь "не просто, а..." — пошел в магазин, заплатил деньги и все! Не хочешь платить наличными — можно сделать то же по перечислению. Правда, здесь есть одна тонкость. Если покупаешь продукт, на котором стоит цена в СКВ (сегодня таких большинство, а в перспективе, похоже, все импортные продукты будут иметь валютную цену), а платишь рублями по курсу, то за время прохождения платежа в банке курс валют может существенно вырасти. В этом случае многие продавцы, чтобы обезопасить себя от скачков на валютной бирже, оговаривают возможность выставления счета на доплату (после получения денег). Правомерность такого подхода с юридической точки зрения представляется достаточно спорной.

Что касается ассортимента товаров, то здесь царит полная неразбериха. Несмотря на все усилия поставщиков, стабильных поставок добиться пока не удается. Причин тому множество, в частности, в этом немалая заслуга уважаемых зарубежных партнеров — фирм-производителей (в том числе самых солидных). Очень сложно также спрогнозировать спрос. Частенько бывает так, что неожиданно приезжают богатые покупатели с периферии и скупают все подчистую. Поэтому сегодня товар есть, завтра, глядишь, уже нет — как говорится, лови момент!

По тем же причинам не балуют нас и особым разнообразием продаваемых товаров. И хотя прайс-листы крупных продавцов пестрят уже сотнями (!) наименований, но ведь в зарубежных-то каталогах программной продукции таких наименований — тысячи! Сюда везут то, что с гарантией можно продать, то есть ширпотреб. Менее же известные (у нас!) продукты пользователям приходится доставать по старинке — в комплекте "без документации".

Итак, список сегодняшних программных бестселлеров — то, что можно купить и что лучше всего покупают на нашем рынке. Список составлен по субъективным оценкам независимых продавцов и экспертов и может быть неполным.

Фирма Microsoft:
MS-DOS 5.0 — операционная система;
Windows 3.1 — графическая операционная система;
Word for Windows 2.0 — текстовый процессор;
Excel 4.0 — электронные таблицы;
Visual Basic 2.0 — средство визуального программирования;
Russian Works 2.0 — интегрированная система;
FoxPro 2.0 — СУБД.

Фирма Borland:
Borland C++ & Application Frameworks 3.1 и Borland Pascal 7.0 — профессиональные компиляторы;
Paradox 4.0 — СУБД;
Paradox Engine & Database Framework 3.0 — библиотека для СУБД;
Quattro Pro 4.0 — электронные таблицы.

Фирма Symantec:
Norton Commander 3.0 — оболочка DOS, русифицированная версия;
Norton DeskTop for Windows 2.0 — пакет утилит для Windows;
Norton Utilities 6.01 — пакет утилит для DOS;
Object Graphics C++ 1.0 и Object Graphics 1.0 for Turbo Pascal for Windows — объектно-ориентированные библиотеки для Windows;
Zortech C++ 3.1 for DOS, Windows, OS/2 — профессиональный компилятор.

Фирма Computer Associates:
Clipper 5.0 — система разработки приложений с поддержкой сетей и баз данных;
Clipper Tools II — библиотека инструментальных средств для Clipper;
Clipper/Compiler Kit — компилятор приложений dBASE IV;
SuperCalc 5.1 и 5.5 — электронные таблицы.

Фирма Lotus:
Lotus 1-2-3 2.3 — электронные таблицы;
cc:Mail — система электронной почты.

Фирма Novell:
DR-DOS 6.0 + NetWare Lite 1.1 — операционная система с поддержкой сети.

Издательские системы и текстовые процессоры:
Page Maker 4.0 фирмы Aldus — издательская система;
WordPerfect 5.1 фирмы WordPerfect — текстовый процессор;
CorelDraw 3.0 фирмы Corel Systems — графический издательский пакет;
Word for Windows 2.0 фирмы Microsoft — текстовый процессор.

По отечественным программным продуктам лидерство упорно держит "народный редактор" Лексикон (фирма Микроинформ) — потрясающая популярность, сравнимая с распространенностью нелегальных копий Norton Commander.

А.Синев



Мультимедиа в трех измерениях

"Крейт" и роль личности в истории

А/О "Крейт" из Санкт-Петербурга — тоже один из зачинателей видеографической революции в стране; для питерского региона фирма сыграла роль, во многом аналогичную роли "Стиплера" в Москве. Но "Крейт" пошел другим путем.

Фирма была создана в 1989 году несколькими выпускниками физмата ЛГУ во главе со Станиславом Амшинским и примкнувшим к ним Антоном Петровым из Военмеха. Антон Петров стал главным стратегом, определяющим техническую политику фирмы — именно он уверенно указал направление, по которому суждено было пойти не только "Крейту", но и — вслед за ним — почти всем петербургским студиям. Это направление было — Amiga. В результате, если в Москве подавляющее большинство студий работает (или начинало) на IBM PC-совместимых компьютерах, то Санкт-Петербург стал колыбелью амиговской технологии — IBM-овские студии там мне и назвать-то затруднялись.

На первоначальный капитал всего в 1,5 тыс. долларов (собранный с помощью салона компьютерных игр и небольших поставок компьютеров) была приобретена первая А500(!) с необходимым минимумом дополнительных плат и софтвера. Сегодня, четыре года спустя, "Крейт" стал солидной фирмой, одним из явных лидеров в области видеоанимации. Помимо собственно производства видеоклипов, "Крейт" занимается поставкой интегрированных видеостудий "под ключ", отдельно компьютеров и видеотехники, а также ремонтом и сервисным обслуживанием вычислительной техники, а также разработкой и производством электронной техники. Фирма имеет представительство в Гамбурге, дочернее агентство в Москве. "Крейт" первым среди наших студий попал в Каталог производителей компьютерной графики PIXEL Corporation.

Основа производственной базы сегодня — 4 компьютера Amiga (2 — A3000 с акселераторами Mercury 68040 и 2 — A4000), с памятью 16 Мбайт и 2 максимально "набитых" IBM PC/486 (по 64 Мбайт RAM), объединенные в сеть (на базе протокола TCP/IP SCO UNIX) со сквозной файловой системой, с распределением не только данных, но и функций. Этот выращенный из маленькой "пятисотки" аппаратно-программный комплекс — предмет особой гордости специалистов "Крейта". Детальную архитектуру комплекса они особо не афишируют — ноу-хау, однако клиентам готовы поставить аналогичный или любое его подмножество, "под ключ". "Для прессы" сообщается лишь, что используется свыше 10 программных пакетов, несколько видеоплат. Впрочем, для зарубежной прессы сделано исключение, и вы сможете почерпнуть перечень основных используемых продуктов из большой статьи о "Крейте" в немецкоязычном журнале "Amiga Plus" (№ 1'93). По этическим соображениям я его воспроизводить не буду, но ничего принципиально нового там нет, практически все используемые платы и пакеты упомянуты в "амиговской" главе нашего обзора — это и не удивительно, основные амиговские инструменты, в общем, достаточно известны. Другое дело, как все эти программы увязаны в цепочки, какие добавлены собственные модули, разработаны контроллеры, как добавить кириллические фонты в генератор символов, и т.д., и т.п. — это-то и есть истинное ноу-хау фирмы. Специалисты "Крейта" постоянно ведут новые разработки, совершенствуют технологии, ищут новые приемы, пробуют "на себе" все новые программы и пакеты для включения их в цепочку. Еще один возможный аспект ноу-хау — это использование малоизвестных "программ одного эффекта", которые во многом уступают конкурентам, но зато могут сделать какое-то одно эффектное преобразование, которого нет в распростра-

Продолжение. Начало в КомпьютерПресс № 1-3,5,6'93

ненных и дорогих пакетах. С помощью такого эффекта можно некоторое время "брать" заказчиков и публику и заставлять конкурентов ломать головы.

Однако держаться все время на передовом уровне — дело не только хлопотное, но и весьма накладное, требующее постоянных немалых вливаний. Поэтому видеонаправление, несмотря на доходы от производства рекламной продукции, дотируется за счет прибылей от остальных видов деятельности "Крейта" — как и было задумано еще при создании студии.

В производстве роликов "Крейт" старается делать упор не на "вал", а на производство качественных клипов со сложными эффектами. Процесс подготовки едва ли не каждого клипа превращается в исследование, в поиск нового — что, с другой стороны, увеличивает время и накладные расходы. Эта вечная коллизия в той или иной форме присутствует в большинстве студий. В "Крейте" ее пытаются решить созданием отдельных групп — производственной и исследовательской. Повышению качества клипов способствует и обозначившаяся "внутренняя творческая конкуренция" между художниками, работающими на амиговской линии и на комплексе PC-Targa, патриоты каждой из линий склонны в хорошем смысле "тянуть одеяло на себя" и выжимать как можно больше именно из своего компьютера.

О качестве роликов "Крейта" лучше всего, пожалуй, говорит эпизод, свидетелем которого я оказался на "ГрафиКон-92". Антон Петров после беседы со специалистом Wavefront (если помните, это одна из ведущих фирм в области 3D-графики, ее пакеты — среди самых дорогих и совершенных инструментов не только на Silicon, но и на Sun и HP), стал показывать ему авторскую кассету роликов "Крейта". Тот, со своих силиконовых высот, взирал на экран спокойно, периодически отпуская какие-то реплики по ходу, пока не спросил, на чем все это сделано. Услышав в ответ, что на Амиге, он был явно поражен, переспросил еще раз, только ли на Амиге, досматривал уже гораздо внимательнее и в заключение, помотав головой, сказал, что это лучшая амиговская графика, которую ему доводилось видеть.

Штат студии "Крейт" сегодня — около 15 человек, в том числе технические специалисты, видеодизайнеры, традиционные художники, сценарист, режиссер, оператор-монтажер, звукорежиссер, композитор. Даже из этого списка видно, что "Крейт" способен обеспечить весь цикл производства клипа; есть для этого и техническая база: магнитофоны Betacam, компьютеризованная тонстудия. Интересно, что видеодизайнеры не имеют специального художественного образования (композитор, математик); попытки же привлечь к этому делу художников-профессионалов пока оканчивались неудачно — у "непрофессионалов" (хотя сегодня их, имеющих двух-трехлетний опыт работы, так назвать можно лишь условно) получается гораздо лучше. Имея в виду, однако, решение этой проблемы в перспективе, "Крейт" организовал семестровый курс компьютерной графики для студентов факультета информационного дизайна Мухинского училища. Сегодня "Крейт", как и большинство лидеров рынка, осваивает Silicon Graphics (с SoftImage); есть станции SGI и в прайс-листах предлагаемого фирмой оборудования. Готовятся и новые шаги в область высоких технологий. Среди задач, решению которых сегодня уделяется наибольшее внимание, — работа для кино, технологии работы с кинопленкой. В этой области свои проблемы, часто весьма отличные от работы с теле- и видеоизображением, — это значительно большее разрешение, отсутствие интерлейсных полей и др. Непросты и задачи перевода изображений с кино на видео и обратно, в компьютер и обратно. Сейчас проблема вывода на кинопленку решается фотографированием экрана с высоким разрешением; вскоре фирма будет располагать и фильм-рекордером. Планируются также работы по совмещению, наложению двух разнородных изображений, а также внутрикадровый монтаж, то есть встраивание сгенерированных компьютером образов в "реальный мир", заснятый на кинопленку; прорабатываются варианты решения этой проблемы и для видеоизображения.

"Крейт" также "активно участвует в общественной жизни": в частности, фирма была спонсором первого из семинаров студии "Пилот", посвященного компьютерной анимации. Сейчас фирма готовится к 3-й Международной Конференции по Компьютерной графике и Визуализации "ГрафиКон-93", которая пройдет в Санкт-Петербурге с 13 по 17 сентября и на которой "Крейт" будет организовывать "Компьютерный видеотеатр". Уже есть договоренности с такими крупнейшими форумами, как Ars Electronica, Imagina, Eurographics, о ретроспективном показе лучших фильмов с этих фестивалей. Все студии страны также приглашаются принять участие, представить свои клипы для показа на "ГрафиКоне" (тел. (812) 311-79-95).

"Послужной список" роликов и заставок у "Крейта" — один из самых больших в стране — он насчитывает более 100 наименований. Это многие заставки Санкт-Петербургского телевидения, оформление "Адамова яблока", "Оранж ТВ", "Большого фестиваля", "Экспресс Кино" (с мчащимся поездом), клипы Кредобанка, Промстройбанка, Банка "Санкт-Петербург", петербургского представительства Philips, SoVenture, A/O "Модерн", "Девиз", собственные ролики "Крейта" и многие другие.

Creat Graphics: путешествие из Петербурга в Москву

В 1991 году Игорь Люско, работавший в Ленинграде в "Крейте", участвовавший в создании многих роликов "Крейта" того периода, переехал в Москву. Он уже был, однако, заражен "амиговским вирусом" и стал пытаться внедрить амиговскую технологию на нашей московской почве, густо поросшей IBM-овскими студиями. С его подачи на цепочке PC-Amiga стало работать рекламное агентство ADMA. Когда же в Москве было создано дочернее предприятие "Крейта", Creat Graphics, Люско стал его руководителем. Две компании

выступают единым фронтом, проводят общую политику, как рыночную, так и техническую, поэтому очень многое из сказанного о "Крейте" относится и к Creat Graphics. Есть, конечно, и свои особенности. В частности, в деятельности Creat Graphics больший упор делается не на собственно производство роликов, а на разработки в области высоких технологий. Фирмой создан и запатентован световой лазерный микроскоп, по разрешению сравнимый с электронным. Кроме того, разрабатываются "железо" для цифрового видеомагнитофона и 24-битная плата с декомпрессией для IBM PC.

Разработка клипов ведется на Amiga-4000 (Impact Vision, Caligary, Imagine, Image Master, ADPro, Vista Pro) и IBM PC (Targa+, TOPAS, 3DS, Animator, TIPS, Diaquest). Список клипов, созданных художниками Creat Graphics, естественно, трудно отделить от списка "Крейта", тем не менее упомянем еще ролики "Kami", а также созданные в сотрудничестве с агентством Adma ролик "Adma" и заставку "Новостей ИТА" первого канала, которую мы наблюдаем ежедневно уже больше года. Фирма сотрудничает с каналами и студиями Останкино, 2x2, RenTV, Megatel. Сейчас вместе с "Крейтом" подготовлены 5 клипов "Погода" RenTV и 6 заставок для 2x2.

"Тройка" мчится...

Это сегодня, пожалуй, наиболее оснащенная студия не только в стране, но и во всей Восточной Европе. Ее историю можно вести с 90-го года, когда Сергей Баженов (ныне — президент "Тройки-Видео") в фирме "ИнтерКонтакт" начал работу с приобретенной через ComputerLand платой Targa+. В 1991 году была организована "Русская Тройка-Видео", которая с самого начала сделала ставку на технику завтрашнего дня, на цифровое видеопроизводство. Были приобретены видеографическая станция Aurora, цифровое запоминающее устройство Abekas, позволяющее накапливать до 2 минут видео профессионального качества, полные аппаратные Betacam, цифровая аудиостудия. О такой мощной видеобазе могли лишь мечтать не только частные студии, но и государственное телевидение. Эта надежная платформа позволяла уверенно заниматься поиском оптимальных средств производства компьютерной графики. (У большинства наших студий, развивавшихся с "компьютерной" стороны, именно видеосоставляющая цепочки создания компьютерного ролика является "узким местом", так как после приобретения компьютеров на оснащение качественной видеотехникой попросту не хватает средств; большинство имеет один-два Betacam'а, а то и вообще арендует время для сброса и монтажа.) В 1992 году были проработаны конфигурации студии на базе Macintosh Quadra и закуплены две Quadra-950 с NuVista, Diaquest DQ-Animaq и MacTOPAS в качестве ведущего 3D-пакета. В конце года "Тройка" приобрела станции Silicon Graphics, и сегодня на наших экранах появляются первые клипы, сделанные на SGI. Технические возможности компании позволяют рассматривать трехмерную компьютерную графику не как основную цель, а как одну из технологий, органичную составную часть нового цифрового телевидения, цифровой обработки и хранения изображений. Сейчас центр компьютерной и цифровой техники "Русской Тройки-Видео" выделился в самостоятельную структуру — B.S.Graphics, также возглавляемую Сергеем Баженовым. Центр приобрел цифровую монтажную станцию HAL фирмы Quantel (182 тыс. ф.ст.), которая позволяет держать в памяти с произвольным доступом и обрабатывать 75 секунд (расширяется до 7,5 мин) видеоизображения broadcast-quality (при этом Quantel не пользуется сжатием — как объясняют представители фирмы, чтобы не терять качества). Изображения хранятся на специальным образом управляемых массивах жестких дисков, так что обеспечивается очень высокая скорость считывания и передачи данных, требуемая для работы с видео в реальном времени. В HAL интегрировано видеографическое устройство PaintBox — пожалуй, самое известное изделие Quantel, работающее во множестве студий по всему миру. PaintBox позволяет вести внутрикадровый монтаж, обработку частей видеоизображения, рисование и ретуширование в кадре. HAL обладает 99 линейными ключами, возможностью создания спецэффектов и работы с цифровым звуком, контроллером VTR.

HAL, Abekas и посты Betacam SP составляют ядро технических средств центра. Для 3D-моделирования, анимации и рендеринга используются две Silicon Graphics Crimson и одна Indigo ELAN. В состав комплекса входит также станция для 2D-анимации и работы с видеоизображением Venice фирмы Jetris Images и еще одно автономное устройство Quantel PaintBox. Quadra-950 теперь оснащены платами MacIvory III английской фирмы Symbolics.

"Тройка", кажется, первой из наших студий компьютерной графики стала работать для зарубежных компаний, на зарубежный экран — для Англии, Испании, Канады, Австрии. Среди клипов для нашего ТВ упомянем РТСБ (летающий брокер), АНА (мальчик с



кистью), РНКБ (с грифоном), Российская Продовольственная Биржа, Кредо-банк (с городом), Олби, Лотто-Миллион ("Я стал миллионером" и "Джек-Пот"), оформление программы НЭП, обе заставки Рекламного агентства "Останкино", рекламу фирм Morline и La Pantera, клип "Ночь на Рождество". Художники студии, в первую очередь Дмитрий Веников, свыше двух лет работали (на некоммерческой основе) над клипом группы "Любэ" "Не валяй дурака, Америка" (или "Алясочка"). В результате клип вобрал в себя элементы разных технологий, в нем использовано множество эффектов, полученных на нескольких компьютерных платформах, на различной видеотехнике; он стал как бы материальным следом эволюции "Тройки" за этот период. В работах студии трехмерная графика соседствует с традиционной анимацией разных стилей, с живыми и преобразованными видеоматериалами, причем все эти изображения не просто перебивают, сменяют друг друга, но могут присутствовать в кадре одновременно, плавно перетекать одно в другое.

B.S.Graphics намерен и в дальнейшем делать ставку на дорогие "high-end" решения: Silicon Graphics в области 3D-графики, Quantel как монтажный и видеографический центр. Уже в этом году предполагается замена (или дооборудование) HAL на станцию HENRY (474 тыс. ф. ст.) — полную цифровую монтажную, вмещающую 15-30 минут видео — по-прежнему без компрессии. Количество станций SGI планируется довести до 5-6, при этом, видимо, кроме используемого сегодня SoftImage, будут применяться и другие пакеты.

Компания контактирует с очень крупными студиями из США и Италии на предмет разработки совместных проектов, производства видеоклипов. При этом западных партнеров привлекает даже не столько уникальная для нашей страны техническая база студии (они оснащены еще лучше), сколько творческий потенциал "Тройки"-B.S.Graphics, идеи, которые режиссеры и художники используют в своих работах. B.S.Graphics в принципе готов заняться и производством компьютерных фильмов — a la Terminator-2, работой в области "цифрового кино". Недавно объявлена система Quantel Domino, позволяющая работать с киноразрешениями 3000x2000, уникальный "черный ящик", у которого вход — отснятые киноматериалы, а выход — готовый смонтированный фильм — также на кинопленке. "Внутри ящика" с оцифрованными изображениями и звуком можно делать воистину все что угодно, а затем сбросить готовую часть фильма прямо на кинопленку. Однако стоимость такой станции — уже около 1,5 млн. ф. ст.!



Хотя основная специализация фирмы не торговая, ее специалисты могут оказать потенциальному клиенту квалифицированную консультацию, подобрать полную конфигурацию студии или отдельную технологическую цепочку в соответствии с требованиями и возможностями заказчика, и поставить ее заказчику "под ключ" — по цене часто более низкой, чем у оригинальных поставщиков.

Hart Studio

Студия Hart образована в 1992 году Андреем Горячевым, Игорем Шичковым и еще несколькими художниками Останкинского телецентра. Фактически это команда профессионалов-"ветеранов", работавшая на Bosch и SuperNova. Hart ставит перед собой чисто творческие задачи: производство роликов и заставок, компьютерная мультипликация, высококачественные синтетические изображения для полиграфии, архитектурный дизайн.

Технический арсенал студии — 5 компьютеров PC/486 с цепочкой Targa-Diaquest, планшеты Kurta, пакеты 3DS, TOPAS, Animator Pro, Image Paint и др.; иногда арендуется и испытанный Bosch. Hart выполняет весь цикл разработки ролика, от идеи и сценария до сброса на видеоленту. Ролики, созданные Hart'ом: Statfal, "Биржа вторичных ресурсов", "Югорский банк" (1), "Трансфрахтсвязь", "Корпорация РИНГ", заставки "Ночное ТВ" (с падающими и прорастающими в манекены стрелами), телекомпании "МИР", заставки открытия и закрытия программ Российского ТВ и др. До прихода в Hart художники студии принимали участие в создании заставок и рекламных клипов 2x2, Театр+TV, 50/50, Всероссийской биржи недвижимости, Всероссийского биржевого банка, Биржи вторичных ресурсов (с бабочкой). В настоящий момент студия рассматривает возможные пути расширения технической базы и склоняется к компьютерам SGI. Hart тесно сотрудничает с Render-club, куда, если вы помните, ушла с ЦТ вторая половина команды с Bosch. Кто знает, может быть Silicon Graphics станет платформой, на которой воссоединится этот коллектив...

* * *

За последний год ряд ведущих студий страны совершили "большой скачок" — прошедший "АниГраф" это окончательно подтвердил — и достигли, помимо про-

чих успехов, такого уровня технической оснащенности, который вряд ли по средствам большинству начинающих команд. Поэтому обзор местами приобретает черты типичной success-story, смотрится несколько оторванным от нашей действительности и в известной степени теряет практическую ценность.

В самом деле, хотя станции Quantel, к примеру, и являются признанным в мире стандартом качества, вряд ли меня поймут, если я, консультируя начинающую студию, посоветую им купить Henry за 800 тысяч долларов... Немногим легче и с Silicon Crimson. Тот же, кто такие средства имеет, как правило, и совета ищет в первоисточнике, предприняв круиз по ведущим студиям мира, по производителям лучшего оборудования.

Поэтому одной из задач нашей рубрики, мне кажется, должно быть рассмотрение более доступных вариантов, поиск конфигураций, которые, возможно, и не в полной мере, но позволят с меньшими затратами решать задачи, для выполнения которых построен Henry или другие very-highend станции. Надо внимательно следить за тенденциями, достижениями технологий мультимедиа, которые могут в очень короткий срок резко изменить ситуацию и уменьшить стоимость оборудования на порядок (вспомните, например, историю видеотостера).

Возможно, скоро мы познакомимся с результатами упомянутой разработки петербургского и московского "Крейтов". В конце мая в Москве прошли презентации компьютерных систем видеомонтажа фирмы Avid Technologies, за которой я с интересом слежу еще с 90-го года. Мы постараемся рассказать о таких системах в одном из ближайших номеров.

Кроме того, мы должны познакомиться с еще несколькими московскими студиями, без которых общая картина осталась бы явно неполной, о ведущих "периферийных" студиях — и поскорее закончить это затянувшееся трехмерное бытописание, пока калейдоскоп не провернулся вновь и наши действующие лица не оказались в новых сочетаниях под новыми флагами...

Наконец, обещанный рассказ о конференции "Мультимедиа: Культура как среда, среда как культура" (февраль, 1993) мы решили несколько отложить. Дело в том, что вслед за ней, как за первым камешком, обрушилась в апреле-мае целая лавина мероприятий "мультимедной" направленности. Поэтому целесообразно будет подождать, пока "пыль уляжется" и взглянуть на эту и последовавшие за ней конференции, выставки и семинары в общем контексте развития мультимедиа в стране.

P.S. Некоторые итоги фестиваля "АниГраф-93", имеющие непосредственное отношение к нашей теме.

Раздел "Компьютерная графика и анимация рекламных клипов"
Приз: студия "Орбис" — рекламный клип чая PickWick — "за лучшее сочетание в использовании средств и приемов компьютерной графики, хорошую идею с использованием элементов национального фольклора, коммерческую направленность с возможностью реальной отдачи от рекламы и краткость в выражении общей идеи".
Дипломы:
А/О "Тивионика" — "Пчелкин Глаз" — ролик "Микроник" — "за хорошую анимацию в сочетании с видеорядом";
Hart Studio — "за общий высокий уровень представленных работ" (представлялись ролики "Трансфрахтсвязь" и "Корпорация РИНГ").
Раздел "Телевизионные заставки"
Приз: заставка "Новой Студии" (автор: Д.Дибров, "Пчелкин Глаз").
Дипломы:
оформление программы "Красный квадрат" (Телекомпания ВИД);
оформление IV канала Останкино ("Пчелкин глаз").
Раздел "Музыкальные клипы"
Приз: студия "Союзмультфильм" (реж. А.Андраникян) — "Цветы" (Валерия).
Диплом: "Русская Тройка-Видео" (прод. С.Баженов, худ. Д.Веников) — "Не валяй дурака, Америка" ("Любэ").

Призы компании Merisel (CorelDraw 3.0) — "за лучший технический уровень" — студии "Русская Тройка-Видео", "Тивионика".
Приз журнала "PC Magazine" — "Микроник" ("Тивионика" — "Пчелкин глаз").

(Подробности — в следующем номере)

С.Новосельцев
Тел.: 237-54-31
Email: next@iplan15.iplan.msk.su

Достойно зарабатывать и заниматься любимым делом

Будучи по личным делам в г.Северодвинске Архангельской области, я обратил внимание на рекламу Справочной Правовой Системы "Консультант плюс" в местной газете. Речь шла о компьютерной библиотеке юридической информации, содержащей нормативные акты законодательных и исполнительных органов власти России с удобными программными средствами для поиска документов и работы с ними. Так как я имел представление о том, что достаточно много московских фирм специализируется на распространении информации такого рода, и она имеет определенный спрос у пользователей, мне стало интересно положение дел в других городах России. Разговор с директором фирмы "Нординсофт" Андреем Валериевичем Росляковым оказался довольно интересным. Вот некоторые выдержки из нашей беседы.

Фирма "Нординсофт" — одна из типичных программистских фирм, — была организована два года назад тремя сотрудниками отдела АСУ крупного государственного предприятия. Наша фирма специализировалась на разработке АРМов под конкретные заказы. Трудности, с которыми мы сталкивались, хорошо знакомы программистам из аналогичных фирм — это сложность поиска заказчика, относительное уменьшение стоимости разработки АРМов, и, как результат, весьма ограниченный размер дохода.

Именно поэтому год назад мы решили расширить свою деятельность и заняться распространением информации. Предварительно провели исследование платежеспособного спроса потребителей различной информации (юридической, биржевой, адресной и пр.). Опрос восьмидесяти фирм показал, что 90% из них заинтересованы, в первую очередь, в юридической информации. Изучив большое число юридических систем, мы остановились на программном продукте "Консультант плюс" фирмы НПО ВМИ (г.Москва).

Решающими факторами выбора послужили, во-первых, финансовые и организационные условия, предложенные НПО ВМИ — минимум начальных вложений, окупаемость в течение двух месяцев, отчисления в НПО ВМИ не более 35% дохода, и во-вторых, оригинальная технология распространения системы "Консультант плюс", реализованная по принципу "запрос-ответ". Такая технология сопровождения выгодно отличается от конкурирующих в этой области систем, использующих механическую замену базы данных пользователя на обновленный вариант. Кроме того, в системе "Консультант плюс" реализована возможность выбора необходимой информации на уровне отдельных документов и оперативного получения новой информации средствами телекоммуникаций. Система не уступает лучшим аналогам по поисковым возможностям и степени сжатия информации на диске, экономящей память компьютера, при этом она достаточно проста и удобна.

Предложенные нам финансовые условия сотрудничества с НПО делали создание и сопровождение базы данных своими силами просто невыгодным, причем результат скорее всего был бы хуже как в смысле информационного наполнения, так, возможно, и в смысле исполнения самой программной оболочки. Итак, мы решили заниматься информационным бизнесом совместно с НПО ВМИ. Начиная это новое для нас дело и не имея практически никакого опыта, мы достаточно скромно оценивали финансовую отдачу от него. Несколько неожиданной оказалась активная сбытовая политика НПО ВМИ и внутренняя убежденность сотрудников НПО, что в нашем небольшом городе мы сможем создать широкую сеть пользователей. Обосновывалось это тем, что распространение системы через регионального представителя является одним из решающих факторов при покупке пользователем системы, обеспечивая ему максимальный сервис при последующем сопровождении.

Сейчас мы убедились, что наши результаты превзошли даже казавшиеся ранее слишком оптимистичными прогнозы НПО ВМИ. В настоящее время, практически не увеличивая штат постоянных сотрудников, мы добились ежемесячного оборота 1.5 млн. руб., из которых 40% составляет наш чистый доход. Более 80 крупнейших предприятий города и области являются нашими постоянными клиентами и ежемесячно их число увеличивается на 8-10.

Сейчас для нас и наших пользователей очевидны преимущества информационной технологии "Консультанта плюс". Главным является простота и полная автоматизация процесса получения новой информации конечным пользователем Системы. Благодаря гибкому механизму обслуживания пользователей нам удалось сделать доход от информационного сопровождения пользователей соизмеримым с доходом от первичных продаж Системы.

Большое количество пользователей Системы позволило нам также постоянно получать заказы на автоматизацию, установку локальных сетей, создание заказных АРМов, то есть все то, что давалось нам раньше с большим трудом.

Не все получалось гладко. Нам потребовалось шесть месяцев напряженной работы, чтобы сейчас достойно зарабатывать и заниматься любимым делом.

С.Таиров

Телефон фирмы "Нординсофт" в Северодвинске 1-94-03.
Телефоны НПО ВМИ (095) 939-43-49, 939-52-15.
E-mail: veda@amcs.msk.su.



Для наиболее эффективного использования "лошадиных сил" вашего компьютера большое значение имеет правильная или, можно сказать, оптимальная конфигурация системы. В частности, в наиболее распространенной сейчас, операционной системе MS-DOS версии 5.0 имеется большое количество драйверов и утилит, которые при правильном, разумеется, использовании могут существенно повысить производительность имеющегося у вас компьютера.

Начнем сначала

Начнем мы, пожалуй, с того, что определим, какие файлы операционной системы MS-DOS версии 5.0 необходимы или могут быть использованы вами с большой вероятностью, а какие без особого ущерба могут быть удалены с жесткого диска вашего компьютера. Поскольку размер "среднего" винчестера в нашей стране по-прежнему не превосходит 40-80 Мбайт, то каждый его свободный килобайт, как говорится, мегабайт бережет. Вообще говоря, наиболее "крутые профи" часто из всей системы оставляют самый минимум: файлы COMMAND.COM, IO.SYS и DOS.SYS, но для среднего пользователя это, пожалуй, не всегда приемлемо.

Итак, если вы не используете переключение кодовых страниц (code page), то без особых раздумий можете удалить с винчестера следующие файлы: DISPLAY.SYS, PRINTER.SYS, EGA.CPI, 4201.CPI, 4208.CPI, 5202.CPI, LCD.CPI, NLSFUNC.EXE и GRAFTABL.COM. Если версия MS-DOS не локализованная, то удалить можно также файлы COUNTRY.SYS, KEYBOARD.SYS и KEYB.COM.

Если вы почитатель J.Socha, который почему-то чаще известен под псевдонимом P.Norton, то без сожаления можете удалить все файлы с именем DOSSHELL.*. За редким исключением могут понадобиться такие файлы, как EGA.SYS и MSHERC.COM, так что, по-видимому, их может ожидать та же участь.

Для компьютеров, совместимых с IBM PC/XT и использующих микропроцессор i8088 (или i8086), никогда не понадобятся файлы HIMEM.SYS, EMM386.EXE и LOADFIX.COM. Для AT´шек, которые, как известно, базируются на микропроцессоре i80286, удалить надо только два последних файла, так как HIMEM.SYS позволит разгрузить стандартную (до 640 Кбайт) память от DOS за счет использования области HMA.

В том случае, если ваш компьютер не оснащен ни expanded-, ни extended-памятью, вам никогда не удастся воспользоваться драйвером SMARTDRV.SYS, поскольку он требует только один из этих двух типов памяти.

Примеры программ, написанные на Бейсике, содержатся в файлах с расширениями *.BAS. Те, кто не признает языка "Ва́сик", могут спокойно удалить их с диска. Сам компилятор (файл QBASIC.COM) можно удалить только в том случае, если вы не применяете текстовый редактор EDIT, который использует оболочку QBASIC.



Наиболее вероятными кандидатами на удаление являются также следующие файлы: APPEND.COM, DISKCOMP.COM, EDLIN.EXE, EXE2BIN.EXE, JOIN.EXE, RECOVER.EXE, REPLACE.EXE и SHARE.EXE.

Поскольку создание файла AUTOEXEC.BAT затруднений обычно не вызывает, то после проведения "экзекуции" по удалению лишних файлов с винчестера можно заняться настройкой файла CONFIG.SYS. В версии 5.0 MS-DOS поддерживается 15 директив (BREAK, BUFFERS, COUNTRY, DEVICE, DEVICEHIGH, DOS, DRIVPARM,

FCBS, FILES, INSTALL, LASTDRIVE, REM, SHELL, STACKS, SWITCHES), которые могут быть включены в этот файл, однако на практике используется, разумеется, только часть из них.

Самой распространенной директивой в файле CONFIG.SYS бывает обычно директива DEVICE. Первым рекомендуемым драйвером, включаемым в файл конфигурации с ее использованием, является HIMEM.SYS. Как мы уже отмечали, речь идет о компьютерах на базе процессоров не ниже 286, имеющих 1 Мбайт (и более) оперативной памяти. Драйвер HIMEM.SYS программно реализует протокол XMS (eXtended Memory Specification), который, в свою очередь, нормализует некоторые правила использования extended-памяти (EMB, Extended Memory Blocks) и памяти в верхних адресах (UMB, Upper Memory Blocks). Кроме этого, спецификация XMS, разумеется, определяет также правила использования области HMA (High Memory Area).

```
DEVICE=C:\DOS\HIMEM.SYS
```

После загрузки HIMEM.SYS появляется возможность разместить модули DOS в области HMA, резидентным программам и драйверам занять свободные UMB, а таким программам как, например, SMARTDRIVE, использовать для своих нужд extended-память. При установке HIMEM.SYS следует иметь в виду, что "старый" способ доступа к extended-памяти через прерывание ROM BIOS int 15h можно обеспечить, задав ключ /INT15=n, где n — количество килобайт extended-памяти, которое предполагается использовать через данное прерывание. Например, если вы хотите использовать 20 Кбайт extended-памяти через прерывание 15h, то строка для драйвера HIMEM.SYS в файле CONFIG.SYS будет выглядеть следующим образом:

```
DEVICE=C:\DOS\HIMEM.SYS /INT15=20
```

Если драйвер HIMEM.SYS необходим практически на всех компьютерах, начиная с AT/286 и тех, что имеют хотя бы 1 Мбайт оперативной памяти, то следующий, не менее полезный драйвер EMM386.EXE можно применять на компьютерах с процессорами не ниже 386SX (что, впрочем, следует и из его названия). Теперь отметим самые существенные моменты. Во-первых, только после установки этого драйвера можно использовать команду LOADHIGH и директиву DEVICEHIGH, которые в MS-DOS версии 5.0 могут загружать в верхнюю UMB-память TSR-программы и драйверы устройств. Во-вторых, именно данный драйвер позволяет преобразовывать всю или некоторое количество extended-памяти в expanded-память, соответствующую спецификации EMS 4.0. Это особенно важно в том случае, когда вы работаете с программами, оперирующими только этим типом памяти. Для того чтобы установить оба эти типа памяти (extended и expanded) строка в CONFIG.SYS должна выглядеть следующим образом:

```
DEVICE=C:\DOS\EMM386.EXE RAM
```

По умолчанию объем expanded-памяти составляет 256 Кбайт. Если такой объем expanded-памяти явно недостаточен, то требуемое ее количество можно указать явно (например, 2 Мбайта):

```
DEVICE=C:\DOS\EMM386.EXE RAM 2048
```

Разумеется, весь указанный объем памяти должен существовать физически. В том случае, когда драйвер EMM386.EXE предполагается использовать только для работы с UMB, то есть без поддержки EMS, строка в CONFIG.SYS должна иметь вид

```
DEVICE=C:\DOS\EMM386.EXE NOEMS
```

Теперь несколько слов о драйвере, который MS-DOS размещает в файле CONFIG.SYS после инсталляции по умолчанию, — это SETVER.EXE. Данный драйвер позволяет использовать прикладные программы, для которых необходима конкретная версия DOS. Если запустить SETVER.EXE как обычную программу из командной строки, то она выдаст список прикладных программ, нуждающихся в поддержке данного драйвера. С большой долей вероятности можно сказать, что рядовой отечественный пользователь не использует ни одну из них, а посему строку, содержащую драйвер SETVER.EXE, из файла CONFIG.SYS можно удалить. В противном (и очень редком) случае данный драйвер лучше разместить в верхней области памяти:

```
DEVICEHIGH=C:\DOS\SETVER.EXE
```

Тем не менее, в некоторых случаях только применение SETVER является единственным способом использования некоторых программ. Например, если вы хотите создать виртуальный диск, используя не XMS, а extended-память (которая, вообще говоря, почти на 64 Кбайта больше), то команда вида

```
SETVER RAMDRIVE.SYS 4.00
```

позволит загрузить драйвер виртуального диска из версии MS-DOS 4.0. Для того чтобы удалить ненужные имена программ из таблицы SETVER, можно воспользоваться следующим выражением

```
SETVER <путь>имя_файла /DELETE
```

Собственно, наверное, уже понятно, что после загрузки драйвера EMM386, работающего с UMB, для освобождения стандартной памяти (до 640 Кбайт) все последующие загружаемые драйверы нужно использовать с директивой DEVICEHIGH. Это в неменьшей степени относится и к следующему драйверу RAMDRIVE.SYS. Правда, заметим, что для компьютера на базе процессора 286 по понятным причинам может использоваться только директива DEVICE.

Драйвер MS-DOS RAMDRIVE.SYS версии 5.0 позволяет создавать виртуальный ("электронный") диск в стандартной, extended- или expanded-памяти, в зави-

симости от указанного в команде ключа: без ключа, /E или /A соответственно. Сразу отметим, что наиболее эффективно использование для данного драйвера extended-памяти.

Часто необходимость применения виртуального диска просто очевидна. Во-первых, он может служить для хранения временных файлов; кстати, при использовании Windows или компилятора Microsoft C — это весьма актуальная задача. В этом случае в файл AUTOEXEC.BAT можно вставить, например, следующую строку:

```
SET TEMP=D:\
```

Отметим, что некоторые другие программы например, LHA, WORD 5.0, "знают" переменную TMP, а не TEMP.

Во-вторых, виртуальный диск хорошо подходит для загрузки программных оверлеев. В-третьих, на таком диске можно хранить часто используемые в процессе работы прикладные программы, предварительно скопировав их с винчестера. Кстати, на виртуальном диске очень удобно хранить и командный процессор:

```
COPY C:\COMMAND.COM D:\
SET COMSPEC=D:\COMMAND.COM
```

Следует помнить, что для MS-DOS версии 5.0 объем "электронного" диска в этом случае должен быть не менее 50 Кбайт (чуть больше размера файла COMMAND.COM).

Таким образом, для создания виртуального диска размером 360 Кбайт в extended-памяти файл CONFIG.SYS надо дополнить следующей строкой:

```
DEVICEHIGH=C:\DOS\RAMDRIVE.SYS
360 /E
```

При использовании для диска expanded-памяти до загрузки драйвера RAMDRIVE.SYS должен быть загружен драйвер поддержки EMS. Например, EMM386.EXE — для компьютеров на базе 386 и 486 процессоров или соответствующий драйвер для системных плат типа NEAT и плат expanded-памяти — для компьютеров на основе процессоров 8088/86 и 286. Кстати, есть одна немаловажная деталь, о которой тем не менее многие часто просто забывают. Она касается максимального числа файлов, которые могут быть записаны в корневой директории виртуального диска. Хотя это значение может варьироваться в пределах от 2 до 1024, но по умолчанию оно составляет всего 64. Об этом следует помнить, если вы хотите сохранить на виртуальном диске (предположим, размером 1 Мбайт) сто файлов, пусть даже каждый из них имеет размер 1 Кбайт. Напомним полный синтаксис команды:

```
DEVICE[HIGH]=[d:][path]RAMDRIVE.SYS [DiskSize]
[SectorSize][NumEntries][/A|/E]
```

Драйвер кэширования диска SMARTDRV.SYS позволяет не только существенно улучшить производительность системы, но и уменьшить общий износ винчестера за счет сокращения количества обращений к нему. Кэш диска может быть организован либо в extended- (без ключа), либо в expanded-памяти (ключ /A). Отметим, что, как и для драйвера RAMDRIVE.SYS, использование extended-памяти в этом случае предпочтительнее. Вообще говоря, чем кэш для жесткого диска больше, тем производительность выше, хотя с определенного объема кэша она растет очень незначительно. Например, для компьютеров с объемом оперативной памяти 4 Мбайта можно рекомендовать добавить следующую строку в файл CONFIG.SYS (напомним, что размер кэш-памяти, образованный эти драйвером, по умолчанию составляет 256 Кбайт):

```
DEVICEHIGH=C:\DOS\SMARTDRV.SYS 512 256
```

Удвоение стандартного размера кэша до 512 Кбайт позволяет таким программам, как Windows, заимствовать в случае необходимости из этой памяти для своих целей 256 Кбайт.

Следует также помнить, что загрузка самого драйвера в верхнюю память (DEVICEHIGH) может окончиться плачевно, если винчестер вашего компьютера оснащен контроллером (ESDI или SCSI), использующим так называемую технику bus mastering и не поддерживающим протокол Virtual DMA Services (VDS) фирмы Microsoft. Как известно, устройство типа bus master может брать управление на себя и для повышения скорости обмена передавать данные непосредственно в оперативную память. Если такое устройство "играет не по правилам", то велика вероятность "повреждения" данных, читаемых или записываемых на винчестер.

К слову сказать, новая версия драйвера SMARTDRIVE, поставляемая вместе с Windows 3.1, позволяет избежать таких проблем благодаря так называемому двойному буферированию. Оно заключается в том, что если даже сам драйвер загружен в верхнюю память, то буферы дорожек все равно хранятся в стандартной памяти.

В заключение напомним, что драйвер SMARTDRV.SYS не поддерживает работу с флоппи-дисками.

Более или менее разобравшись с директивой DEVICE (DEVICEHIGH) и драйверами, используемыми чаще всего, перейдем теперь к рассмотрению возможностей других директив, включаемых в файл CONFIG.SYS.

Директива DOS для компьютеров на базе 286 процессоров (при наличии не менее 1 Мбайта оперативной памяти) должна записываться следующим образом:

```
DOS=HIGH
```

Это позволяет перенести модули DOS в установленную ранее область HMA и освободить около 60 Кбайт стандартной памяти. Заметим, что по умолчанию (то есть если такая строка отсутствует в файле конфигурации) — DOS=LOW, что соответствует загрузке системы

в стандартную память, даже если HIMEM.SYS был загружен. Для компьютеров на базе процессоров 386 и выше та же строка может выглядеть следующим образом:

```
DOS=HIGH,UMB
```

Это, в частности, означает, что DOS, используя драйвер HIMEM.SYS, объявляет все блоки UMB, открытые EMM386.EXE, для своего применения. Заметим также, что по умолчанию DOS=NOUMB.

Как известно, MS-DOS воспринимает нажатие комбинаций клавиш Ctrl-C и Ctrl-Break так, что пользователь хочет прекратить выполнение текущей операции. Однако по умолчанию DOS проверяет нажатие этих клавиш только в случае записи символа на экран, последовательный порт или принтер, а также когда читает символ или состояние клавиатуры. Следующее выражение, добавленное в файл CONFIG.SYS, позволяет расширить выполняемые проверки, включив в них, в частности, чтение и запись файлов на диск:

```
BREAK=ON
```

Очевидным недостатком при этом является существенное замедление операций с файлами, так что, если вы не имеете достаточно веских причин для прерывания исполняемых программ, то подобную строку в CONFIG.SYS лучше не использовать.

В файле CONFIG.SYS могут применяться две директивы, которые определяют количество одновременно используемых файлов: FCBS и FILES. Начиная с версии 2.0 MS-DOS предлагает для прикладных программ новое средство для работы с файлами — так называемые хэндлы файлов (file handles). Это значит, что прикладная программа, открывая файл, получает некий уникальный номер, который в дальнейшем может использоваться для работы с этим файлом. Вообще говоря, хэндл файла является индексом в таблице, где DOS хранит необходимую информацию о файле. Синтаксис этой директивы

```
FILES=n
```

где n определяет максимальное количество файлов, которые DOS может поддерживать одновременно. По умолчанию n=8, а при необходимости может варьироваться в пределах от 8 до 255. Некоторые приложения требуют 20 и более одновременно открытых файлов. Это касается, в частности, баз данных и сетевых программ. Наиболее резонным значением в общем случае (при отсутствии конкретных указаний) является все же число 20. Не следует забывать, что на каждый предусмотренный файл требуется 59 байт стандартной памяти.

Структура, которая может ставиться в соответствие каждому открытому файлу (она, кстати, была заимствована из CP/M), носит название *блок управления файлом* (File Control Block, FCB). Таким образом, директива FCBS определяет, как много файлов может быть открыто с использованием FCB. По умолчанию это значение равно 4, но может изменяться в пределах от 1 до 255.

Как известно, сигналы от различных периферийных устройств, таких например, как клавиатура, последовательный или параллельный порт, могут прерывать работу процессора — это аппаратные прерывания. Причем возникать они могут с частотой от 20 до нескольких сотен раз в секунду. Эти прерывания обрабатываются специальными сервисными программами ISR (Interrupt Service Routine), которые, в частности, должны сохранить текущее состояние критичных регистров процессора и адрес возврата, чтобы после обработки вернуться в прерванную программу и правильно восстановить содержимое регистров. Понятно, если аппаратные прерывания следуют очень часто, то стек небольшой длины может переполниться. Именно для этой цели используется набор специальных внутренних стеков.

Директива STACKS определяет количество таких стеков в наборе и их размер, например:

```
STACKS=8,256
```

Здесь, в частности, определено 8 внутренних стеков размером 256 байт каждый, а в и без того небольшой стандартной памяти можно "вычеркнуть" еще 2 Кбайта. По умолчанию для компьютеров на базе процессоров 8088/86 используется значение STACKS=0,0, а на базе процессоров 286 и выше — STACKS=9,128. На вопрос, какие значения в данной директиве необходимы для конкретного компьютера, ответить, как правило, трудно. Попробуйте начать со значения STACKS=0,0, в худшем случае система "рухнет", сообщив вам напоследок примерно следующее:

```
Internal stack overflow
System halted
```

Для ускорения обмена с дисками DOS организует в оперативной памяти специальные внутренние буферы, которые работают примерно так же, как дисковый кэш. Количество буферов устанавливается в CONFIG.SYS директивой вроде

```
BUFFERS=20
```

Каждый буфер занимает в оперативной памяти 512 байт, то есть равен размеру дискового сектора. Прямой зависимости между количеством буферов и производительностью дисковых операций, откровенно говоря, не существует. Наиболее широко распространено мнение о том, что чем больше буферов — тем лучше, но больше 50 — уже хуже. Некоторые считают, что объем буферов должен быть не меньше объема FAT. Другие полагают, что количество буферов следует делать кратным количеству секторов на дорожку диска. Никак не комментируя все это, отмечу лишь, что MS-DOS 5.0 способна разместить в HMA до 47 буферов.

Директива BUFFERS определяет не только кэш первого уровня, эффективный при работе с диском, когда к нему осуществляется произвольный (случайный) доступ, но и кэш второго уровня, особенно необходимый при последовательной записи и чтении файлов. При использовании же "настоящего" кэширования (SMARTDRIVE) количество буферов обычно рекомендуется предельно минимизировать — то есть до 2, а кэш второго уровня и вовсе не указывать.

По умолчанию количество буферов первого уровня составляет от 2 до 15, в зависимости от объема стан-

дартной памяти и типов приводов, установленных в системе. Количество буферов второго уровня обычно не превышает 4.

Когда в файле CONFIG.SYS указана директива DOS=HIGH, буфера автоматически перемещаются также в область HMA. Для каждого буфера требуется приблизительно 532 байта памяти — размер сектора плюс заголовок. Кстати, если в компьютере отсутствует expanded- и extended-память, то есть нет возможности установить драйвер для кэша диска, то "по Малинину-Буренину" количество буферов первого уровня можно определить как половину емкости винчестера в мегабайтах. Следовательно, при наличии жесткого диска в 60 Мбайт можно записать следующее выражение:

```
BUFFERS=30,4
```

Для каждого логического диска, определяемого в системе, MS-DOS выделяет 88 байт стандартной памяти для хранения информации об этом устройстве. По умолчанию DOS резервирует пространство для пяти дисков (от A до E), включая и логические.

```
LASTDRIVE=E
```

Директива LASTDRIVE позволяет отвести место для хранения информации о 26 дисках, созданных в том числе и командой SUBST или сетевыми утилитами:

```
LASTDRIVE=Z
```

Для экономии стандартной памяти директива LASTDRIVE, используемая в конкретном файле CONFIG.SYS, должна определять последний реально существующий диск.

Директива INSTALL позволяет включить в файл конфигурации резидентные утилиты DOS, например, FASTOPEN.EXE. Этот способ загрузки экономит, в частности, 160 байт (или больше), расходуемых обычно на блок памяти для хранения копии программной среды (environment).

Кстати, утилита FASTOPEN.EXE является своеобразным кэшем диска, который, в отличие от уже упомянутого драйвера SMARTDRV.SYS, работает не на уровне секторов, а на уровне файлов. В том случае, когда отсутствует extended- или expanded-память, то единственной возможностью ускорить доступ к наиболее часто используемым файлам является именно эта утилита. FASTOPEN.EXE кэширует информацию о физическом расположении открываемых файлов на диске, что позволяет вторично открыть их гораздо быстрее.

Данная утилита поддерживает работу с 24 логическими дисками (от C до Z), параметр, указываемый после буквы, обозначает количество файлов, информация о которых может кэшироваться. Например:

```
INSTALL=C:\DOS\FASTOPEN.EXE C:=20 D:=10
```

то есть в данном случае для диска C могут кэшироваться данные о 20 файлах, а для D — о 10. Заметим, что информация о каждом файле, хранимая в специальном буфере, требует 48 байт памяти. Практически, обычно считают, что при использовании FASTOPEN необходимо открыть хотя бы один буфер на каждый мегабайт дискового пространства. Так, если логический диск C имеет объем 40 Мбайт, а D — 20 Мбайт, то справедливо следующее выражение:

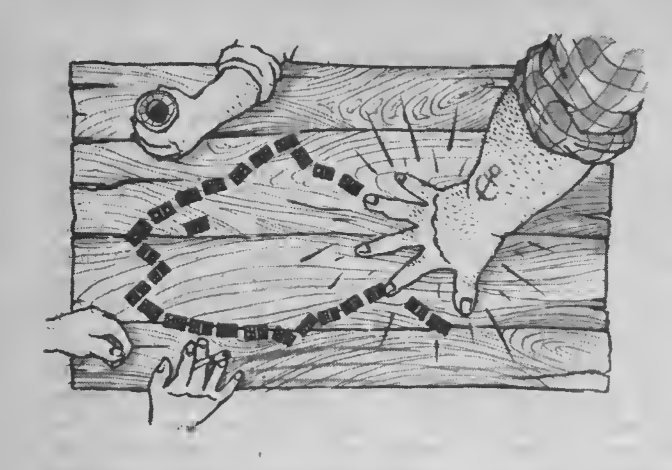
```
INSTALL=C:\DOS\FASTOPEN C:=(40,20) D:=20 /X
```

Дополнительный числовой параметр кэширования для диска C: позволит ускорить доступ к фрагментированным файлам, запоминая местонахождение непрерывных областей файлов и каталогов. Ключ /X позволяет использовать для этой утилиты expanded-память. В случае если таковая отсутствует, можно преобразовать имеющуюся extended- в требуемую expanded-память при помощи драйвера EMM386.EXE или загружать FASTOPEN.EXE из файла AUTOEXEC.BAT в верхнюю память командой LOADHIGH.

На этом мы закончим рассмотрение наиболее распространенных директив и драйверов, включаемых обычно в файл CONFIG.SYS MS-DOS версии 5.0. Надеемся, что у читателей не будет претензий к тому, что мы не рассмотрели хорошо "заезженный" драйвер ANSI.SYS и т.п. В заключение приведем пример файла CONFIG.SYS для компьютера на базе процессора 386 (или 486), который имеет 4 Мбайта оперативной памяти и винчестер емкостью 120 Мбайт, поделенный на три логических диска C (20 Мбайт), D (40 Мбайт) и D (60 Мбайт):

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE RAM 2048
DEVICEHIGH=C:\DOS\RAMDRIVE.SYS 360 /E
DEVICEHIGH=C:\DOS\SMARTDRV.SYS 512
256
DOS=HIGH,UMB
FILES=20
BUFFERS=2
LASTDRIVE=F
```

А.Борзенко



В этом месяце среди множества игр, появившихся, чуть было не написал "на нашем рынке...", мне приглянулись три, на которых мы и остановимся.

Новые игры

Comanche — Maximum Overkill
Nova Logic

Хотя я и не люблю все эти авиационные симуляторы (наверное сказывается опыт службы в ВВС), эта игра привлекла мое внимание чрезвычайно высоким качеством исполнения. Для ее работы требуется 386 процессор и 4 Мбайта памяти. На 33 МГц вертолет летает вполне пристойно, а сама игра умещается на 12 Мбайтах. Вылеты происходят в холмисто-гористой местности, что, с одной стороны, затрудняет маневры, а с другой — делает игру более интересной. Для уничтожения противника имеются пушки, стингеры и другие различные эффективные средства. По мере прохождения этапов игры и увеличения опыта сражаться становится все труднее, враги становятся все коварнее. Для полного кайфа можно использовать джойстик (сейчас их полно в магазинах по 20-40 тысяч). Игра Comanche — достойное пополнение коллекции авиасимуляторов (для тех, кто их коллекционирует).

Sleep Walker
Ocean Software

Игра Sleep Walker, выполненная в стиле мультипликации, посвящена несчастному мальчику, который ходит во сне (и куда смотрят родители!). Все бы закончилось трагедией, не будь рядом верного пса. Собака постоянно должна направлять движение мальчика. На эту тему можно сложить "черный стишок":

Ма́ленький мальчик ходит во сне
.... размазались по стене.

Отлично выполненная, легко управляемая, эта игра может подойти даже маленьким детям.

Rex Nebular and the Cosmic gender-bender
MicroProse

Приключения капитана Рекса, в классическом стиле quest'ов — это для тех, кто любит приключенческие игры. Звездный капитан отправился куда-то за чем-то (см. вводный мультфильм). Где-то в глубинах галактики патрульный корабль подбил его, и он не совсем мягко приземлился на планете амазонок (это такие слегка одетые крутые накачанные девушки с бластерами). Дальше — надо лазить по планете, куда нас занесло, собирать что-то, что-то использовать, разговаривать, короче говоря — типичный quest. Игра выполнена в разрешении 320x200, что слегка ухудшает ее внешний вид. Интерфейс — то, что неофициально называется a-la LucasArts: внизу приводится список возможных команд. Удобным является то, что для каждого подобранного предмета приводится список его свойств: дохлую рыбу можно понюхать (фу-у) или ... съесть, таймер можно разобрать, в бинокль — посмотреть и т.д. Имеется два варианта интерфейса — простой (это когда по нажатию кнопки мыши выполняется команда Look) и сложный — когда команда по умолчанию не выполняется. Кстати об интерфейсах. На мой взгляд, такой интерфейс является наиболее разумным. Использовавшийся ранее в играх Sierra интерфейс с командной строкой требовал знания английского языка, а современный — чрезвычайно упрощен. А еще мне нравится, когда поместив курсор мыши на какой-то объект, можно получить его название.

На планете есть ведьмы, амазонки, различные сверхсовременные технические объекты — если станете играть, несколько нескучных вечеров вам обеспечено. Те, кто владеет языком (английским), могут оценить юмор создателей, который иногда находится на грани.

Буквально на днях появилось продолжение Eco Quest фирмы Sierra — Lost Secret of the Rainforest. Выполненная в стиле последних игр — Twisty History и Turbo Science, эта игра относится к серии обучающих игр — гуляя по джунглям, мы можем узнать много интересного о флоре, фауне и истории этого края. Как всегда, мягкий юмор и различные приколы не на последнем месте. Игра поставляется как для MS-DOS, так и для Windows, но в последнем исполнении работает достаточно медленно. В настоящий момент я набрал 150 очков из 1000 возможных и более подробно расскажу о ней в следующий раз.

На будущей неделе обещали принести продолжение игры Kyrandia, следите за публикациями!

А.Федоров

НОВОСТИ

2 июня 1993 года в рамках IV Международного Компьютерного форума состоялась совместная пресс-конференция фирм IBS и Dell Europe о стратегии Dell Corporation в России.

Как известно, эта американская компания является одним из трех крупнейших мировых производителей персональных компьютеров. За прошлый финансовый год рост корпорации Dell составил 126%. Этот фантастический результат стал возможен благодаря новому подходу к разработкам и поставкам машин, а также превосходному сервису и поддержке клиентов.

На пресс-конференции выступили Генеральный Директор фирмы IBS г-н Карачинский и Директор Dell по Восточной Европе г-н Левицкий. Фирмы подписали партнерский договор, по которому выработкой стратегии на российском рынке будет отныне заниматься компания IBS. К наиболее важным задачам маркетинговой политики были отнесены такие, как создание независимой дилерской сети, разумные цены при высоком качестве, надежное сервисное обслуживание, и все это, разумеется, с учетом требований, предъявляемых Dell. Будем надеяться, что слова, сказанные однажды Майклом Деллом, "клиент не может быть просто удовлетворен, клиент должен быть счастлив", будут скоро справедливы и в России.

Новости от Hewlett-Packard

Нынешний 1993 год для московского представительства Hewlett-Packard знаменателен тем, что исполняется 25 лет присутствия фирмы на российском рынке. Заметим, что если раньше львиную долю в общем объеме продаж на отечественном рынке составляло периферийное оборудование, то сейчас наметились явные сдвиги к изменению этой ситуации. Так, если в 1991 году доля персональных компьютеров составила всего 5% от общего объема продаж, в 1992 — 12-15%, то в этом году она ожидается на уровне 20-25%. Последняя цифра, по словам представителей фирмы, соответствует среднеевропейскому уровню.

Как известно, Hewlett-Packard уделяет огромное внимание созданию новых моделей лазерных и струйных принтеров. Например, программа развития струйных принтеров была принята компанией еще в 1979 году. По некоторым прогнозам доля лазерных и струйных принтеров от Hewlett-Packard на рынке в этом году может достигнуть 50 и 52% соответственно.

Новый струйный принтер **HP DeskJet 510** заменит устаревшую модель DeskJet 500. Основные улучшения коснулись скорости печати и механизма подачи бумаги. Так, печать в качественном режиме LQ (Letter Quality) выполняется на 40% быстрее, а в лоток для автоматической подачи бумаги можно загружать теперь до 100 листов или 20 конвертов. В режиме LQ максимальная разрешающая способность принтера составляет 300 точек на дюйм, а скорость печати — 3 страницы в минуту. Фирма гарантирует безотказную гарантию устройства в течение 3 лет. Ожидаемая цена для российского рынка объявлена на уровне 480 долл.

Модель цветного струйного принтера **HP DeskJet 1200C** не является продолжением какой-либо существующей серии, а открывает новую. Данный принтер способен печатать в трех режимах: нормальном, высококачественном и скоростном. Заметим, что при работе в черно-белом режиме с использованием технологии RET разрешающая способность может составлять 600x300 точек на дюйм, а скорость печати — 7 страниц в минуту. В цветном режиме производительность и разрешающая способность несколько снижаются — 1 страница в минуту и 300 точек на дюйм соответственно. На сегодняшний день это, пожалуй, самый быстрый из цветных струйных принтеров во всем мире. Интересной особенностью нового принтера можно считать то, что он может использовать те же платы расширения памяти и кассеты со шрифтами, что и модель лазерного принтера LaserJet 4. Поскольку в модели DeskJet 1200C используются те же 45 шрифтов, что и в LaserJet 4, то любой документ, подготовленный на одном из них, можно смело распечатывать на другом.

Помимо основной модели будет выпущена и ее модификация — DeskJet 1200C/PS. Эта модель в первую очередь отличается наличием язы-

ков управления PCL5 и PostScript (Level 2). Оперативная память данного принтера может расширяться до 20 Мбайт. Стандартный лоток для подачи позволяет хранить до 180 листов формата А4.

Новая модель лазерного принтера **LaserJet 4L** основана на микропроцессоре Motorola 68000, имеет разрешающую способность 300 точек на дюйм (использует технологию RET), скорость печати составляет до 4 страниц в минуту. В новом принтере применена новая технология управления памятью (MET, Memory Enhancement Technology), использующая режим виртуальных страниц и позволяющая практически удваивать реальный имеющийся объем памяти. Встроенный режим экономичной печати (Econom Mode) может снизить стоимость одной печатной страницы почти вдвое.

Наиболее привлекательными для пользователя могут оказаться цена (949 долларов) нового принтера и его размеры (362x353x164 мм).

Четвертый МКФ

Со 2 по 4 июня в Московском Центре Международной торговли на Красной Пресне проходил Четвертый Международный компьютерный форум (МКФ). Вот уже четвертый год подряд он проводится Международным компьютерным клубом (МКК) — неправительственной международной общественной организацией, созданной в декабре 1988 г. Ее членами являются более 170 зарубежных и российских фирм, таких как Hewlett-Packard, Bull, Zeos International, Lotus Development, Borland International, Symantec, Pick Systems, Aldus, Steepler, Интермикро, Merisel, Интермикро Бизнес Системы и другие.

В рамках Четвертого МКФ прошла выставка современных информационных технологий, в которой участвовали Zeos International, RUI Apple Computer IMC, Borland, IBM, Интермикро Бизнес Системы, Интермикро, Steepler, НПО "Нилстар", АйТи, Summit Systems, Hewlett-Packard, Демос+APS-COM, Dell Computer, Logitech, Intel, National Instruments.

Здесь впервые в мире были продемонстрированы новые продукты Zeos International — компьютеры субблокнотного формата Zeos Contenda на базе процессора Intel 486SL-25, а также новый цветной компьютер-блокнот Zeos Colornote 486SX-33. На стенде Zeos был показан и один из самых быстрых в мире (по результатам многочисленных тестовых испытаний) персональных компьютеров — Zeos DX2-66.

Разработчики НПО "Нилстар" представили оригинальный одноплатный IBM-совместимый компьютер, все детали и компоненты которого, за исключением интеловского процессора, изготовлены в России.

Как всегда, отличительной чертой очередного форума МКК был высокий уровень и "представительный" состав выступающих. Среди них: Грег Херрик, Председатель Правления Zeos International (США), Рик Инатом, Председатель Правления InaCom (США), Эстер Дайсон, Президент EDventure Holdings (США), Роберт Финоккио, Вице-Президент 3Com Corporation (США), Роберт Митзе, Управляющий Директор UNIX System Laboratories (Англия), Скотт Хансен, Управляющий Директор UNIX International (Англия), Дэвид Уиттл, Руководитель группы OS/2, IBM (США), Фрэд Ланга, Директор Издательского Совета Windows Magazine и др.

Выставку посетило около 30 тысяч специалистов, представляющих все структуры власти, а также специалисты и разработчики из многих стран СНГ.

ГрафиКон'93

С 13 по 17 сентября в Санкт-Петербурге под традиционным девизом "Компьютерная графика в науке и искусстве" пройдет очередная (уже третья) международная конференция ГрафиКон'93. Конференция организуется Научно-техническим и творческим обществом "ГРАФО", Институтом точной механики и оптики при содействии РАН, американского общества компьютерной графики ACM SIGGRAPH, IEEE и европейского общества EUROGRAPHICS.

Предыдущие конференции вызвали большой интерес, на них выступили многие всемирно известные специалисты в области компьютерной графики. В этом году будут традиционные выставки компьютерного искусства и современных технологий компьютерной графики, образовательные семинары, компьютерный видеотеатр, круглые столы и многое другое.

Попасть на конференцию просто — звоните в Оргкомитет по телефону (812) 238-85-24. Также можно послать сообщение электронной почтой: G93@impmo.spb.su G93@keldysh.msk.su.

Новый "ПК для всех"

Небольшое сообщение для всех любителей персональных компьютеров: недавно вышел первый номер нового ежемесячного журнала для программистов и пользователей "ПК для всех". Пока его можно получить бесплатно: желающие могут прислать письмо по адресу 394030 Воронеж, а/я 81. Не забудьте вложить в письмо конверт с вашим адресом.

О русской защите замолвлено слово

Начал разрушаться миф о странах, где программы не копируются. В ноябре на выставке COMDEX'92 в США распространялась подборка материалов из европейских и американских изданий о судебных делах против воровства программ. Воруют везде. Самые разные случаи, от громких судебных процессов над компьютерными пиратами, принесшими миллионные убытки, до курьезов. Отказу от применения средств защиты послужила, очевидно, не только вера во всемогущество и справедливость западных законов, но и начатая примерно семь лет назад усиленная кампания в прессе против использования защиты от копирования. Применяемые в то время системы грешили своим качеством и доставляли пользователям массу неудобств.

В последнее время на мировом рынке систем защиты ситуация стала меняться. Яркий тому пример — случай с ивритской версией Windows, поставлять которую Microsoft хотела было с защитой от копирования, но из-за отсутствия последней все же выпустила на рынок незащищенную версию. Западный рынок оказался не готов к такому обороту...

Российский рынок защит: два года назад — 37 систем (из них 23 были представлены на SofTool'91), прошлый год — 26 (на SofTool'92 представлено 8), из которых заслуживают внимания 16 (см. статью "16 вариантов русской защиты", КомпьютерПресс 10'92). В нынешнем году со сцены сошли еще несколько фирм, хотя, судя по количеству публикаций в компьютерной прессе, интерес к проблемам защиты не только не уменьшился, а многократно возрос.

Долгая изнурительная борьба разработчиков систем защиты с изощренными отечественными "взломщиками" заставляла искать и реализовывать новые идеи и алгоритмы, поднимая планку требований все выше и выше. И вот эта борьба увенчалась первыми успехами — по результатам тестирования, проведенного израильской фирмой OLAN Advanced Technologies & Software совместно с EliaShim Microcomputers лучшей признана русская защита File_PROTECTION фирмы NOVEX Software, получившая наивысшие оценки по всем ключевым показателям: совместимость с "железом" и различным программным обеспечением, стойкость, удобство работы и простота установки защиты.

Совместными усилиями трех фирм подготовлена промышленная версия системы File_PROTECTION, ориентированная на крупные заводы — производители программного обеспечения, которой с апреля 1993 года начали оснащаться производители ПО в Нидерландах, Греции, Израиле, Италии, Франции, США, Португалии, Испании, Чехословакии и Южной Африке. Продаваться за рубежом новый File_PROTECTION будет как File_PROTECTION System, а также под известной торговой маркой фирмы EliaShim — CODESAFE с индексом NT (New Technology).

Другие известные системы защиты фирмы NOVEX Software — генератор дистрибутивных дискет FP_Installator, системный драйвер и комплект библиотек для динамического шифрования данных Lock_MANAGER — также не были оставлены без внимания. Права на них недавно приобрела польская фирма Highland Systems International. На выставке ЭКСПОКОМ-93 большой интерес вызвала система NOVEX Netware NAVIGATOR, предназначенная для подготовки, рекламирования и продажи защищенного от копирования программного обеспечения с использованием средств телекоммуникаций.

DELL + ЛЕКСИКОН

Новая формула для новой России, и вывели ее сообща три компании: Dell Computer (США), Intermicro Business Systems (Россия) и МИКРОИНФОРМ (Россия). Суть соглашения, подписанного с участием перечисленных фирм, и девиз которого вынесен в заглавие данной заметки, состоит в следующем — отныне ВСЕ поставляемые в Россию компьютеры DELL будут комплектоваться текстовым процессором ЛЕКСИКОН.

Корпорация Dell, основанная Майклом Деллом в 1984 году, является в настоящий момент третьей мировой величиной по производству персональных компьютеров (после IBM и Compaq). Что же касается *качества* продукции, то теперь имеются все основания считать продукцию Dell первой в мире. Об этом говорят результаты 16 независимых исследований предпочтения пользователей и оценки около 150 компьютерных экспертов за 1992 год.

Выходя на российский рынок, фирма Dell стремилась прежде всего не изменить своим правилам, ставя во главу угла все тот же обеспечивший ей мировой успех беспрецедентный уровень сервиса. Но для полной поддержки фирменного "делловского" комплекса услуг, избавляющего клиента от в с е х проблем, американской фирме был необходим опытный и сильный деловой партнер на территории России. Как известно, им стала фирма IBS, выделившаяся из СП Интермикро в 1992 году.

Закономерным продолжением политики Dell, цель которой — *дать пользователю то, что он хочет*, было решение IBS устанавливать на компьютеры программное обеспечение, не ограничиваясь при этом стандартным набором из MS-DOS и Windows. Каким требованиям обязан удовлетворять такой software? Это должна быть очень популярная, хорошо зарекомендовавшая себя программа *базисного* характера, требующаяся пользователю ежедневно, ежечасно. С учетом специфики рынка это должен быть инструмент с непременной российской локализацией и отлаженным механизмом технической поддержки фирмой-изготовителем.

И единственным программным продуктом, просто не имеющим конкурентов по всем перечисленным параметрам, оказался текстовый процессор ЛЕКСИКОН! Созданный Е.Н.Веселовым в 1985 году, ЛЕКСИКОН приобрел массовую популярность в России еще до 1991 года, когда поддержкой и продвижением этой программы на рынке занялась фирма МИКРОИНФОРМ. К успехам фирмы, кроме неуклонно растущего объема продаж ЛЕКСИКОНа на территории России, следует отнести выход на европейский и американский рынок. Международному успеху редактора безусловно способствует поддержка набора и печати шрифтами национальных европейских алфавитов, не последнюю роль играет и постоянно совершенствующийся пользовательский интерфейс.

С компьютерами Dell будет поставляться новейшая на сегодняшний день версия редактора — ЛЕКСИКОН 1.2 (апрель 1993 года), причем стоимость машин НЕ будет увеличиваться за счет предустановки программы. Естественно, новоиспеченные владельцы компьютеров Dell станут легальными пользователями ЛЕКСИКОНа и смогут пользоваться всеми видами сервиса и технической поддержки фирмы МИКРОИНФОРМ, включая право на скидку при upgrade по выходе новой версии (которая ожидается не далее как осенью 1993 года). Настоящее соглашение будет действовать до конца 1993 года, оговорены основные направления дальнейшей совместной деятельности МИКРОИНФОРМ и Dell.

До сей поры, несмотря на все усилия, МИКРОИНФОРМу удалось заключить лишь одно подобное соглашение о сотрудничестве с производителем аппаратного обеспечения, а именно с фирмой IBM, причем об установке программ на *каждый* компьютер речь не шла. Что же касается отечественных OEM, то до них, как ни странно, "достучаться" оказалось вообще невозможно. Видимо, у нас мало кого заботит поддержка и защита интересов российских производителей программных продуктов.

На этом фоне очень радует цивилизованный подход и серьезность намерений по отношению к российскому рынку фирмы столь высокого уровня, как Dell.

ПараГраф стал ближе к МУРу — удачно подметил кто-то из собравшихся на журналистский "круглый стол", организованный российско-американским СП ПараГраф — одной из крупнейших наших программистских фирм — в своем новом офисе, расположенном в непосредственной близости с Петровкой, 38.

Организаторы встречи рассказали собравшимся об успехах и дальнейших планах фирмы. Как водится, воплощение новых идей и разработка передовых технологий идут рука об руку с успехами в бизнесе. Лицензии на шрифты библиотеки ParaType приобретены крупнейшими западными фирмами, в числе которых Microsoft и крупнейший в Европе продавец шрифтов — FontShop GmbH. Ведется разработка кириллических шрифтов для всемирно известной библиотеки фирмы ITC по лицензионному соглашению с последней.

Своеобразный обмен любезностями состоялся между ПараГрафом и фирмой Adobe. Разработчик языка PostScript теперь лицензировал ParaWin, пакет русификации Windows, а ПараГраф приобрел права на Adobe Type Manager.

Работы по русификации западных программных продуктов вступили в новую фазу. Кроме новой локализации MS Word (будет продаваться связка Word 5.5 — Русское Слово 3.0 — Орфо 3.0), журналистам продемонстрировали работу самого средства перевода системы меню и сообщений Word 5.5 на любой язык — пакета WordMaker.

Исследования ПараГрафа в области multimedia — особая статья. "Электронный секретарь" с синтезатором речи — пример одностороннего "мостика" между символьной и речевой media, объяснил на пресс-конференции Леонид Малков. Одним из "мостиков" между "бумажной" и символьной формой представления информации являются системы OCR, поэтому хотелось бы ожидать интересных результатов от недавно начатых партнерских отношений ПараГрафа и фирмы "Бастион", разработчика системы распознавания текста TIGER.

К.Ахметов, А.Борзенко, С.Груздев

Индекс 73217